Key Rotation for Authenticated Encryption

Adam Everspaugh[1], Kenneth Paterson[2], Thomas Ristenpart[3], Sam Scott[2]

[1]University of Wisconsin–Madison, [2]Royal Holloway, University of London,
[3]Cornell Tech

Abstract. A common requirement in practice is to periodically rotate the keys used to encrypt stored data. Systems used by Amazon and Google do so using a hybrid encryption technique which is eminently practical but has unclear security in the face of key compromises and does not provide full key rotation. Meanwhile, symmetric updatable encryption schemes (introduced by Boneh *et al.*, CRYPTO 2013) support full key rotation without performing decryption: ciphertexts created under one key can be rotated to ciphertexts created under a different key with the help of a re-encryption token. By design, the tokens do not leak information about keys or plaintexts and so can be given to storage providers without compromising security. But the prior work of Boneh *et al.* addresses relatively weak confidentiality goals and does not consider integrity at all. Moreover, as we show, a subtle issue with their concrete scheme obviates a security proof even for confidentiality against passive attacks.

This paper presents a systematic study of *updatable authenticated encryption*. We provide a set of security notions that strengthen those in prior work. These notions enable us to tease out real-world security requirements of different strengths and build schemes that satisfy them efficiently. We show that the hybrid approach currently used in industry achieves relatively weak forms of confidentiality and integrity, but can be modified at low cost to meet our stronger confidentiality and integrity goals. This leads to a practical scheme that has negligible overhead beyond conventional AE. We then introduce *re-encryption indistinguishability*, a security notion that formally captures the idea of fully refreshing keys upon rotation. We show how to repair the scheme of Boneh *et al.*, attaining our stronger confidentiality notion. We also show how to extend the scheme to provide integrity, and we prove that it meets our re-encryption indistinguishability notion. Finally, we discuss how to instantiate our scheme efficiently using off-the-shelf cryptographic components (AE, hashing, elliptic curves). We report on the performance of a prototype implementation, showing that fully secure key rotations can be performed at a throughput of approximately 118 kB/s.

The original version of this work that appeared at Advances in Cryptology – Crypto 2017 had a number of flaws. While the majority of the core concepts remain unaltered, this version constitutes a significant rewrite. We highlight the most significant changes with footnotes throughout the paper.

We thank Joseph Jaeger for initially pointing out bugs in various proofs, and for substantial advice with regards to recovering positive results. We also thank Anja Lehmann for making us aware of issues with the XOR-KEM construction and limitations of our bidirectional security notions. We have removed the XOR-KEM construction, which was falsely claimed in the previous version to meet our bidirectional security notion.

1 Introduction

To cryptographically protect data while stored, systems use authenticated encryption (AE) schemes that provide strong message confidentiality as well as ciphertext integrity. The latter prevents active attackers who manipulate ciphertexts. When data is stored for long periods of time, good key management practice dictates that systems must support key rotation: moving encrypted data from an old key to a fresh one. Indeed, key rotation is mandated by regulation in some contexts, such as the payment card industry data security standard (PCI DSS) that dictates how credit card data must be secured [PCI16]. Key rotation can also be used to revoke old keys that are compromised, or to effect data access revocation.

Deployed approaches to key rotation. Systems used in practice typically support a type of key rotation using a symmetric key hierarchy. Amazon's Key Management Service [AWS], for example, enables users to encrypt a plaintext m under a fresh data encapsulation key x via $C = \mathcal{E}(x, m)$ and then wrap x via $\tilde{C} = \mathcal{E}(k, x)$ under a long-term key k owned by the client. Here $\mathcal{E}$ is an authenticated encryption (AE) scheme and the final ciphertext is $(\tilde{C}, C)$. By analogy with the use of hybrid encryption in the asymmetric setting, we refer to such a scheme as a KEM/DEM construction, with KEM and DEM standing for key and data encapsulation mechanisms, respectively; we refer to the specific scheme as AE-hybrid.

The AE-hybrid scheme allows a simple form of key rotation: the client picks a fresh k' and re-encrypts x as $\tilde{C}' = \mathcal{E}(k', \mathcal{D}(k, \tilde{C}))$. Note that the DEM key x does not change during key rotation. When deployed in a remote storage system, a client can perform key rotation just by fetching from storage the small, constant-sized ciphertext $\tilde{C}$, operating locally on it to produce $\tilde{C}'$, and then sending $\tilde{C}'$ back to the storage system. Performance is independent of the actual message length. The Google Cloud Platform [Goo] uses a similar approach to enable key rotation.

To our knowledge, the level of security provided by this widely deployed AE-hybrid scheme has never been investigated, let alone formally defined in a security model motivated by real-world considerations. It is even arguable whether AE-hybrid truly rotates keys, since the DEM key does not change. Certainly it is unclear what security is provided if key compromises occur, one of the main motivations for using such an approach in the first place. On the other hand, the scheme is fast and requires only limited data transfer between the client and the data store, and appears to be sufficient to meet current regulatory requirements.

Updatable encryption. Boneh, Lewi, Montgomery, and Raghunathan (BLMR) [BLMR15] (the full version of [BLMR13]) introduced another approach to enabling key rotation that they call *updatable encryption*. An updatable encryption scheme is a symmetric encryption scheme that, in addition to the usual triple of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ algorithms, comes with a pair of algorithms $\mathsf{ReKeyGen}$ and ReEnc. The first, $\mathsf{ReKeyGen}$, generates a compact rekey token given the old and new secret keys and a target ciphertext, while the second, ReEnc, uses a rekey token output by the first to rotate the ciphertext without performing decryption. For example, AE-hybrid can be seen as an instance of an updatable encryption scheme in which the rekey token output by $\mathsf{ReKeyGen}$ is $\tilde{C}'$ and where ReEnc simply replaces $\tilde{C}$ with $\tilde{C}'$. BLMR introduced an IND-CPA-style security notion in which adversaries can additionally obtain some rekey tokens. Their definition is inspired by, but different from, those used for CCA-secure proxy re-encryption schemes [CH07]. Given its obvious limitations when it comes to key rotation, it is perhaps surprising that the AE-hybrid construction provably meets the BLMR confidentiality notion for updatable encryption schemes.

BLMR also introduced and targeted a second security notion for updatable encryption, called ciphertext independence. This demands that a ciphertext and its rotation to another key are identically distributed to a ciphertext and a rotation of another ciphertext (for the same message). The intuition is that this captures the idea that true key rotation should refresh all randomness used during encryption. This definition is *not* met by the AE-hybrid construction above. But it is both unclear what attacks are prevented by meeting their definition, and whether more intuitive definitions exist.

BLMR gave a construction for an updatable encryption scheme and claimed that it provably meets their two security definitions. Their construction cleverly combines an IND-CPA KEM with a DEM that uses a key-homomorphic PRF [NPR99,BLMR15] to realize a stream cipher. This enables rotation of both the KEM and the DEM keys, though the latter requires a number of operations that is linear in the plaintext length. Looking ahead, their proof sketch has a bug and we provide strong evidence that it is unlikely to be fixable. Moreover, BLMR do not target or achieve any kind of authenticated encryption goal.

Scheme	Section	UP-IND	UP-INT	UP-REENC
AE-hybrid†	2	✗	✗	✗
KSS*	6	✓	✓	✗
BLMR	8	✗	✗	✗
ReCrypt*	9	✓	✓	✓

Table 1. Summary of schemes studied. † In-use by practitioners today. * Introduced in this work.

Our contributions. We provide a systematic treatment of AE schemes that support key rotation without decryption, what we refer to as *updatable AE*. Specifically, we provide a new security notion for confidentiality, UP-IND, that is strictly stronger than that of BLMR. We also provide a security notion for ciphertext integrity, UP-INT, and finally a new notion called re-encryption indistinguishability (UP-REENC). The latter is stronger than the ciphertext indistinguishability notion of BLMR and, we believe, more intuitively captures the spirit of "true key rotation".

Achieving our UP-REENC notion implies that an attacker, having access to both a ciphertext and the secret key used to generate it, should not be able to derive any information that helps it attack a rotation of that ciphertext. Thus, for example, an insider with access to the encryption keys at some point in time but who is then excluded from the system cannot make use of the old keys to learn anything useful once key rotation has been carried out on the ciphertexts. Teasing out the correct form of this notion turns out to be a significant challenge in our work.

Armed with this set of security notions, we go on to make better sense of the landscape of constructions for updatable AE schemes. Table 1 summarizes the security properties of the different schemes that we consider. Referring to this table, our security notions highlight some limitations of the AE-hybrid scheme: we prove it meets confidentiality UP-IND and UP-INT security, but only when considering an adversary who has no access to any compromised keys. As discussed in the body, one can additionally show that AE-hybrid, even in the presence of compromised keys, can achieve a weakened version of our UP-IND notion. In either case the construction does not achieve a satisfying level of security, and so we propose an improved construction called KSS. It satisfies both UP-IND and UP-INT for any number of compromised keys and is easily deployable via small adjustments to AE-hybrid. KSS uses a form of secret sharing to embed key shares in the KEM and DEM components to avoid the issue of leaking the DEM key in the updating process, and has the KEM encrypt the DEM ciphertext's authentication tag to prevent ciphertext integrity attacks. These changes could easily be adopted by practitioners with virtually no impact on performance, while significantly improving security.

However, the improved scheme KSS cannot satisfy our UP-REENC notion, because it still uses a KEM/DEM-style approach in which the DEM key is never rotated. The BLMR scheme seems a natural starting point for a stronger scheme. We start by showing, unfortunately, that proving the BLMR scheme confidential would imply that one could also prove circular security [BRS03,CL01] for a particular type of hybrid encryption scheme assuming only the key encapsulation is IND-CPA secure. Existing counter-examples of IND-CPA secure, but circular insecure, schemes [ABBC10,CGH12] do not quite rule out such a result. But the link to the very strong notion of circular security casts doubt that one can provide a formal security analysis of this scheme. One can easily modify the BLMR scheme to avoid this issue, but even having done so the resulting encryption scheme is still trivially malleable and so cannot meet our UP-INT integrity notion.

We therefore provide another new scheme, ReCrypt, meeting all three of our security notions: UP-IND, UP-INT and UP-REENC. We take inspiration from the previous constructions, especially that of BLMR: key-homomorphic PRFs provide the ability to fully rotate encryption keys; our KEM/DEM approach with secret sharing avoids the issue of leaking the DEM key in the updating process; and finally, including an authentication tag within the KEM plaintext tightly binds the KEM and DEM portions and prevents ciphertext manipulation.

We go on to instantiate the scheme using the random oracle model (ROM) key-homomorphic PRF from [NPR99]. This yields a construction of an updatable AE scheme meeting all three of our security notions in the ROM under the DDH assumption. We report on the performance of an implementation of ReCrypt using elliptic curve groups, concluding that it is performant enough for practical use with short

plaintexts. However, because of its reliance on relatively expensive elliptic curve operations, ReCrypt is still orders of magnitude slower than our KSS scheme (that achieves only UP-IND and UP-INT security). This, currently, is the price that must be paid for true key rotation in updatable encryption.

Summary. In summary, the main contributions of this paper are:

- To provide the first definitions of security for AE supporting key rotation without exposing plaintext.
- To explain the gap between existing, deployed schemes using the KEM/DEM approach and "full" refreshing of ciphertexts.
- To provide the first proofs of security for AE schemes using the KEM/DEM approach, namely AE-hybrid and KSS.
- To detail the first updatable AE scheme, ReCrypt, that fully and securely refreshes ciphertexts by way of key rotations without exposing plaintext. We implement a prototype and report on microbenchmarks, showing that rotations can be performed in less than $10\,\mu$s per byte.

2 Preliminaries

Authenticated encryption. A symmetric encryption scheme π is a tuple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. $\mathcal{K}$ is a randomized algorithm, called key generation, that outputs a secret key (a bit string of some fixed length). We let $\mathbb{K}$, called the key space, be the support of $\mathcal{K}$ (the bit strings with probability greater than zero of being output by $\mathcal{K}$). For a bit string m called the message, we denote by $C \leftarrow\!\!\$\, \mathcal{E}_k(m)$ the randomized algorithm for encryption using key k and producing a bit string C, called the ciphertext. By $\mathcal{D}_k(C)$ we denote the (deterministic) decryption algorithm. Decryption either outputs a message or a distinguished error symbol $\perp$.

An authenticated encryption (AE) scheme is, formally, just a symmetric encryption scheme. Informally we will use AE to refer to schemes meant to provide both confidentiality and integrity, as formalized below. For AE schemes we will also abuse notation slightly and sometimes have ciphertexts, the output of $\mathcal{E}_k(m)$, be a pair of bit strings (C, τ) (instead of a monolithic single string C). We refer to C as a ciphertext portion and τ as an authentication tag or simply tag. The latter facilitates explicit handling of τ. Unless otherwise stated, our encryption schemes will be length-regular, meaning that the lengths of ciphertexts depend only on the lengths of plaintexts. This ensures that the real-or-random notions given below imply the more standard "left-or-right" indistinguishability notions. We associate to any AE scheme π a function $\mathcal{CS}_\pi$ that maps a length to the set of all ciphertexts for plaintexts of that length. For example, if a scheme has ciphertexts that consist of any $|m|+n$-bit string and any β-bit tag, then $\mathcal{CS}_\pi(|m|) = \{0,1\}^{|m|+n} \times \{0,1\}^\beta$. When clear from context we will drop the subscript π.

We require that AE schemes are correct, meaning that for all messages and secret keys, it holds that $\Pr[\mathcal{D}_k(\mathcal{E}_k(m)) = m] = 1$, where the probability is over the coins used by encryption.

We focus here on randomized encryption (rather than nonce-based [Rog04]) and dispense with associated data for simplicity. We believe our results extend to other settings.

AE security notions. To facilitate our proofs, we present multi-user variants of standard security notions for symmetric encryption. The multi-user aspect refers to the fact that an adversary can interact with a number of independent instances of the encryption scheme. We use for AE security a multi-user variant of the all-in-one notion from [RS06]. (Our notation here mostly follows the code-based framework of [BR06].) Let $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme. Games MU-ROR-AE1$_\pi$ and MU-ROR-AE0$_\pi$ are detailed in Figure 1. The table $\mathtt{C}$ is intialized to $\perp$ everywhere (this is true of all tables used in games in this paper). The parameter t controls the number of encryption instances the adversary can interact with while trying to infer bit b. Notice that we have deviated slightly from traditional AE security notions [RS06] in that we use, in the MU-ROR-AE0$_\pi$ case a table of returned ciphertexts in order to have decryption correctly reply to these values. This will be convenient when using the notion within reductions, and is equivalent (for correct AE schemes) to the more traditional approach that has decryption always output $\perp$. The advantage of an MU-ROR-AE$_\pi$-adversary $\mathcal{A}$ is measured by:

$$\mathsf{Adv}^{\text{mu-ror-ae}}_\pi(\mathcal{A}) = \left|\Pr\left[\,\text{MU-ROR-AE1}^{\mathcal{A}}_\pi \Rightarrow 1\,\right] - \Pr\left[\,\text{MU-ROR-AE0}^{\mathcal{A}}_\pi \Rightarrow 1\,\right]\right| .$$

The probabilities here (and in other similar probability statements involving games) is taken over the coins used in executing the game.

MU-ROR-AE1$^{\mathcal{A}}_{\pi,t}$	Enc(i,m)	Dec(i,C)
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{Enc,Dec}}$	If $\mathtt{K}[i] = \bot$ then	If $\mathtt{K}[i] = \bot$ then
Return b'	$\quad k_i \leftarrow\!\!\$\ \mathcal{K}$	$\quad k_i \leftarrow\!\!\$\ \mathcal{K}$
	$C \leftarrow\!\!\$\ \mathcal{E}(k_i, m)$	Return $\mathcal{D}(k_i, C)$
	Return C	

MU-ROR-AE0$^{\mathcal{A}}_{\pi,t}$	Enc(i,m)	Dec(i,C)
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{Enc,Dec}}$	$C \leftarrow\!\!\$\ \mathcal{CS}(\lvert m\rvert)$	Return $\mathtt{C}[i,C]$
Return b'	$\mathtt{C}[i,C] \leftarrow m$	
	Return C	

Fig. 1. Multi-user, real-or-random security games for AE scheme $\pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

We also will use a chosen-plaintext-only variant of MU-ROR-AE, in which the attacker is disallowed from making decryption queries. Let game MU-ROR1$^{\mathcal{A}}_{\pi}$ (resp. MU-ROR0$^{\mathcal{A}}_{\pi}$) be the same as MU-ROR-AE1$^{\mathcal{A}}_{\pi}$ (resp. MU-ROR-AE0$^{\mathcal{A}}_{\pi}$) except that the decryption oracles are removed. We define the ROR$_\pi$ advantage of $\mathcal{A}$ as:

$$\mathsf{Adv}^{\text{mu-ror}}_{\pi}(\mathcal{A}) = \left|\Pr\left[\,\text{MU-ROR1}^{\mathcal{A}}_{\pi} \Rightarrow 1\,\right] - \Pr\left[\,\text{MU-ROR0}^{\mathcal{A}}_{\pi} \Rightarrow 1\,\right]\right| .$$

Finally, as some extra notation we will let $\mathsf{Adv}^{\text{mu-ror-ae}}_{\pi,t}(\mathcal{A})$ denote advantage when $\mathcal{A}$ is limited to querying t distinct indices to its oracles. We call such an adversary an MU-ROR-AE$_{\pi,t}$ adversary. Also in the chosen-plaintext setting, we let $\mathsf{Adv}^{\text{ot-ror}}_{\pi}(\mathcal{A})$ denote advantage of an adversary $\mathcal{A}$ that can query its encryption oracle on each index at most once (i.e., "one-time" security). We call such an adversary an OT-ROR$_\pi$ adversary.

PRFs and key-homomorphic PRFs. Let $\mathcal{K}, \mathcal{X}, \mathcal{Y}$ be sets. Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be an efficiently computable function. Informally we say that F is a good PRF if it is indistinguishable from a random function. More formally, let game PRF1$^{\mathcal{A}}_{F}$ be the game that selects a key $k \leftarrow\!\!\$\ \mathcal{K}$ and then runs an adversary $\mathcal{A}$ that can adaptively query to an oracle that returns F_k applied to the queried message. The adversary outputs a bit. Let game PRF0$^{\mathcal{A}}_{F}$ be the game in which an adversary $\mathcal{A}$ can adaptively query an oracle that returns a random draw from $\mathcal{Y}$. The adversary outputs a bit. We assume that $\mathcal{A}$, in either game, never queries the same value twice to its oracle. We define the PRF$_F$ advantage of $\mathcal{A}$ as:

$$\mathsf{Adv}^{\text{prf}}_{F}(\mathcal{A}) = \left|\Pr\left[\text{PRF1}^{\mathcal{A}}_{F} \Rightarrow 1\right] - \Pr\left[\text{PRF0}^{\mathcal{A}}_{F} \Rightarrow 1\right]\right| .$$

We will also use a multi-use PRF security variant. It is implied by the standard PRF notion just given. Formalizing this variant will be convenient in some of our proofs. In detail, let MU-PRF1$_F$ be the game that gives an adversary $\mathcal{A}$ access to an oracle that, upon being queried a pair i, x for i a number and $x \in \mathcal{X}$, it first determines if i has previously been queried, if not, it generates a new secret key $k_i \leftarrow\!\!\$\ \mathcal{K}$, then returns $F_{k_i}(x)$. The adversary outputs a bit, which the game also outputs. Let MU-PRF0$_F$ be the game that gives an adversary $\mathcal{A}$ access to an oracle that, upon being queried an index i and value $x \in \mathcal{X}$, samples a fresh value $y \leftarrow\!\!\$\ \mathcal{Y}$ and returns it. The adversary outputs a bit, which the game also outputs. The adversary cannot repeat (i, x) queries (but can query the same x under different i values). We then define the MU-PRF$_F$ advantage of an adversary $\mathcal{A}$ as:

$$\mathsf{Adv}^{\text{mu-prf}}_{F}(\mathcal{A}) = \left|\Pr\left[\,\text{MU-PRF1}^{\mathcal{A}}_{F} \Rightarrow 1\,\right] - \Pr\left[\,\text{MU-PRF0}^{\mathcal{A}}_{F} \Rightarrow 1\,\right]\right| .$$

Associate to $\mathcal{K}$ and $\mathcal{Y}$ operations, such that $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are groups, where we have abused notation for the sake of simplicity and used the same additive operator for both groups (though, in fact, they will be distinct in instantiations). In either group, we write $k \cdot x$ for $k \in \mathbb{Z}$ and x in the group to represent adding x to itself k times. A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is called key-homomorphic if for every $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, it holds that

$$F(k_1, x) + F(k_2, x) = F(k_1 + k_2, x) .$$

Informally, a key-homomorphic PRF is a key-homomorphic function that is secure in the sense of being a PRF. A simple example of a secure key-homomorphic PRF in the ROM is the function $F(k, x) = k \cdot H(x)$ where H maps $\mathcal{X}$ to $\mathcal{Y}$ and $(\mathcal{Y}, +)$ is an additive group in which the decisional Diffie–Hellman assumption holds. This construction is originally due to Naor, Pinkas, and Reingold [NPR99].

Resource usage conventions. Our theorem statements will reason about the resources used by adversaries in terms of oracle queries and run time. The latter will be measured in a model where pseudocode statements

are unit cost and queries are unit cost. We will often write that some adversaries (used in a reduction) run in time equal to that of another adversary plus some $\mathcal{O}(n)$ overhead. This means that the adversary's worst-case run time is at most that of the underlying adversary plus $c \cdot n$ operations for some small constant c. The exact constant can be derived from the proof. We therefore use big-O notation to hide unimportant constants.

3 Updatable AE

We start by formalizing the syntax and semantics of AE schemes supporting key rotation. Our approach extends that of Boneh et al. [BLMR15] (BLMR), the main syntactical difference being that we allow rekey token generation, re-encryption, and decryption to all return a distinguished error symbol $\perp$. The last is required when considering schemes that reject ciphertexts, a requisite for ciphertext integrity. We also modify the syntax so that ciphertexts include two portions, a header and a body. In our formulation, only the former is used during generation of rekey tokens (while in BLMR the full ciphertext is formally required). This allows us to more accurately capture the behavior of schemes in which only a portion of the ciphertext is needed in order to generate rekey tokens.

Updatable AE schemes. Formally an *updatable AE scheme* is a tuple of algorithms $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Dec})$.

- *Key generation:* The randomized key generation algorithm outputs a secret key k, which we denote by $k \leftarrow\!\!\$\ \mathsf{KeyGen}()$.
- *Encryption:* On input a secret key k and message m, the randomized encryption algorithm outputs a ciphertext $(\tilde{C}, C)$ consisting of a ciphertext header $\tilde{C}$ and ciphertext body C. We denote this by $(\tilde{C}, C) \leftarrow\!\!\$\ \mathsf{Enc}(k, m)$.
- *Rekey generation:* On input two secret keys, k_1 and k_2, and a ciphertext header $\tilde{C}$, the randomized rekey token generation algorithm outputs a rekey token or a distinguished error symbol $\perp$. We denote rekey generation by $\Delta_{1,2,\tilde{C}} \leftarrow\!\!\$\ \mathsf{ReKeyGen}(k_1, k_2, \tilde{C})$.
- *Re-encryption:* On input a rekey token and ciphertext, the re-encryption algorithm outputs a new ciphertext or $\perp$. We require that ReEnc is deterministic. We denote re-encryption by $(\tilde{C}', C') \leftarrow \mathsf{ReEnc}(\Delta_{1,2,\tilde{C}}, (\tilde{C}, C))$.
- *Decryption:* On input a secret key k and ciphertext $(\tilde{C}, C)$, the deterministic decryption algorithm outputs either a message or $\perp$. We denote this by $m \leftarrow \mathsf{Dec}(k, (\tilde{C}, C))$.

Of course we require that all algorithms are efficiently computable. Note that, in common with [BLMR15], our definition is *not* in the nonce-based setting that is widely used for AE. Rather, we will assume that Enc is randomized. We consider this sufficient for a first treatment of updatable AE; it also reflects common industry practice as per the schemes currently used by Amazon [AWS] and Google [Goo]. Similarly, we assume that all our AE schemes have single decryption errors [BDPS14], and we do not consider issues including release of unverified plaintext [ABL+14], tidiness [NRS14] and length-hiding [PRS11]. We defer developing parallel formulations capturing these other settings and issues to future work.

Correctness. An updatable AE scheme is *correct* if decrypting a legitimately generated ciphertext reproduces the original message. Legitimate ciphertexts may be rotated through many keys, complicating the formalization of this notion.

Definition 1 (Correctness). *Fix an updatable AE scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Dec})$. For any message m and any sequence of $T \geq 1$ secret keys $k_1, \ldots, k_T$ output by running KeyGen, let $(\tilde{C}_1, C_1) \leftarrow\!\!\$\ \mathsf{Enc}(k_1, m)$ and recursively define for $1 \leq t < T$ the ciphertexts $(\tilde{C}_{t+1}, C_{t+1}) = \mathsf{ReEnc}(\mathsf{ReKeyGen}(k_t, k_{t+1}, \tilde{C}_t), (\tilde{C}_t, C_t))$. Then Π is correct if $\mathsf{Dec}(k_T, (\tilde{C}_T, C_T)) = m$ with probability one, that probability being over the randomness used by Enc and $\mathsf{ReKeyGen}$.*

Compactness. We say that an updatable AE scheme is *compact* if the size of both ciphertext headers and rekeying tokens are independent of the length of the plaintext. In practice the sizes should be as small as possible, and for the constructions we consider these are typically a small constant multiple of the key length.

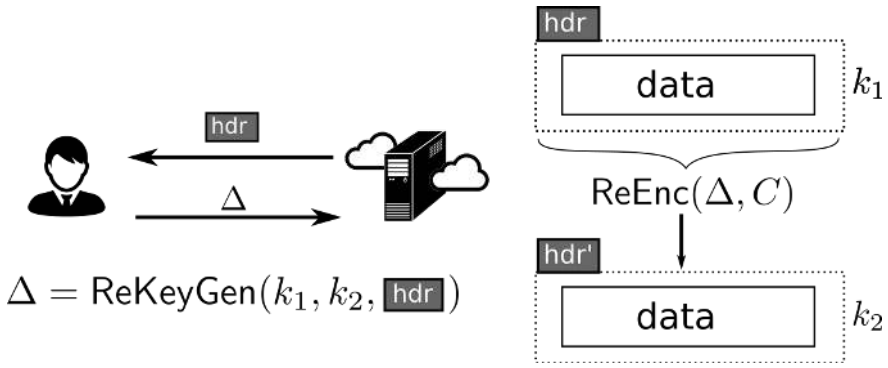

Fig. 2. Interaction between client and remote storage service during a ciphertext-dependent update. Client retrieves a small ciphertext header, and runs ReKeyGen to produce a compact rekey token Δ. The storage service uses this token to re-encrypt the data. At the end of the update, the data is encrypted using k_2, and cannot be recovered using only k_1.

Compactness is important for efficiency of key rotation. Considering the abstract architecture in Figure 2, ciphertext headers must be available to the client when rekey tokens are generated. Typically this will mean having to fetch them from storage. Likewise, rekey tokens must be sent back to the storage system. Note that there are simple constructions that are not compact, such as the one that sets $\tilde{C}$ to be a standard authenticated encryption of the message and in which ReKeyGen decrypts $\tilde{C}$, re-encrypts it, and outputs a "rekeying token" as the new ciphertext.

Ciphertext-dependence. As formulated above, updatable AE schemes require part of the ciphertext, the ciphertext header $\tilde{C}$, in order to generate a rekey token. We can also consider schemes for which $\tilde{C}$ is the empty string, denoted ε. We will restrict attention to schemes for which encryption either always outputs $\tilde{C} = \varepsilon$ or never does. In the former case we call the scheme *ciphertext-independent* and, in the latter case, *ciphertext-dependent*. When discussing ciphertext-independent schemes, we will drop $\tilde{C}$ from notation, e.g., writing C for the ciphertext instead of $(\tilde{C}, C)$ and writing $\Delta_{i,j}$ instead of $\Delta_{i,j,\tilde{C}}$.

While we discuss ciphertext-independent schemes briefly, our primary focus will be on ciphertext-dependent schemes.[1]

Directionality of rotations. Some updatable AE schemes are *bidirectional*, meaning that a single rekey token can be used to rotate a ciphertext from a key k_1 to a key k_2 *and* vice versa. For simplicity we only consider bi-directionality to be a feature of ciphertext-independent schemes. Formally, we say that a scheme is *bidirectional* if there exists an efficient algorithm $\mathsf{Invert}(\cdot)$ that produces a valid rekey token $\Delta_{j,i}$ when given $\Delta_{i,j}$ as input.

Schemes that are not bidirectional might be able to ensure that an adversary cannot use rekey tokens to "undo" a rotation of a ciphertext. We will see that ciphertext-dependence can help in building such unidirectional schemes, whereas ciphertext-independent schemes seem harder to make unidirectional. This latter difficulty is related to the long-standing problem of constructing unidirectional proxy re-encryption schemes in the public-key setting.

Relationship to proxy re-encryption. Proxy re-encryption targets a different setting than updatable encryption (or AE): the functional ability to allow a ciphertext encrypted under one key to be converted to a ciphertext decryptable by another key. The conversion should not leak plaintext data, but, unlike key rotation, it is not necessarily a goal of proxy re-encryption to remove all dependency on the original key, formalized as indistinguishability of re-encryptions (UP-REENC security) in our work. For example, previous work [CK05,ID03] suggests double encryption of plaintexts under different keys. To rotate, the previous outer key and a freshly generated outer key are sent to the proxy to perform conversion, but the inner key is never modified. Such an approach does not satisfy the goals of key rotation.

That said, any bidirectional, ciphertext-independent updatable AE scheme ends up also being usable as a symmetric proxy re-encryption scheme (at least as formalized by [BLMR15]).

[1] In an earlier version of this paper, we introduced a ciphertext-independent scheme, called XOR-KEM. However our prior claims about its security are incorrect.

$\underline{\text{UP-IND}^{\mathcal{A}}_{\Pi,\kappa,t}}$
$b \leftarrow\!\!\$\ \{0,1\}$
$k_1, \ldots, k_{t+\kappa} \leftarrow\!\!\$\ \mathsf{KeyGen}()$
$b' \leftarrow\!\!\$\ \mathcal{A}^{O}(k_{t+1}, \ldots, k_{t+\kappa})$
Return $(b' = b)$

$\underline{\text{LR}(i, m_0, m_1)}$
If $\mathsf{Mal}(i)$ or $|m_0| \neq |m_1|$ then
  Return $\perp$
$(\tilde{C}, C) \leftarrow\!\!\$\ \mathsf{Enc}(k_i, m_b)$
$\mathtt{T}[i, \tilde{C}] \leftarrow C$
Return $(\tilde{C}, C)$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j)$ and $\mathtt{T}[i, \tilde{C}] \neq \perp$ then
  Return $\perp$
$\Delta_{i,j,\tilde{C}} \leftarrow\!\!\$\ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
If $\mathtt{T}[i, \tilde{C}] \neq \perp$ then
  $(\tilde{C}', C') \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, \mathtt{T}[i, \tilde{C}]))$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow C'$
Return $\Delta_{i,j,\tilde{C}}$

$\underline{\text{Enc}(i, m)}$
Return $\mathsf{Enc}(k_i, m)$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, C))}$
$\Delta_{i,j,\tilde{C}} \leftarrow\!\!\$\ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
$(\tilde{C}', C') \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, C))$
If $\mathsf{Mal}(j)$ and $\mathtt{T}[i, \tilde{C}] \neq \perp$ then
  Return $\tilde{C}'$
If $\mathtt{T}[i, \tilde{C}] \neq \perp$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow C'$
Return $(\tilde{C}', C')$

Fig. 3. Confidentiality security game for updatable encryption security.

4 Confidentiality and Integrity for Updatable Encryption

Updatable AE should provide confidentiality for messages as well as integrity of ciphertexts, even in the face of adversaries that obtain rekey tokens, re-encryptions, and some number of corrupted secret keys. Finding definitions that rule out trivial wins — e.g., rotating a challenge ciphertext to a compromised key, or obtaining sequences of rekey tokens that allow such rotations — is delicate. We provide a framework for doing so.

Our starting point will be a confidentiality notion UP-IND which improves upon the previous notion of BLMR by modeling additional attack vectors. For ciphertext integrity, we develop a new definition, UP-INT, that builds on the usual INT-CTXT notion for standard AE [BN00]. Looking ahead, we will target unidirectional schemes that simultaneously achieve both UP-IND and UP-INT security.

We will follow a concrete security approach in which we do not strictly define security, but rather measure advantage as a function of the resources (running time and number of queries) made by an adversary. Informally speaking, schemes are considered secure if no adversary with reasonable resources can achieve advantage far from zero.

4.1 Message Confidentiality for Updatable AE Schemes

The confidentiality game $\text{UP-IND}_{\Pi,\kappa,t}$ is shown in Figure 3. It is parameterized by an updatable AE scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Dec})$, two numbers $t > 0$ and $\kappa \geq 0$, and an adversary $\mathcal{A}$. The game initializes $t + \kappa$ secret keys, κ of which are given to the adversary. The keys $k_1, \ldots, k_t$ are the uncompromised keys and the keys $k_{t+1}, \ldots k_{t+\kappa}$ are the compromised keys. We stress that the indices here are only used to track keys in our security definitions and do not refer to any temporal ordering or usage of the keys. We let $\mathsf{Hon}(i) = (i \leq t)$, i.e., the key is honest, and let $\mathsf{Mal}(i) = (i > t)$, i.e., the key is compromised. We require at least one uncompromised key ($t > 0$), but do not necessarily require any compromised keys ($\kappa \geq 0$). The adversary's goal is to guess a random challenge bit b, this bit being used to determine which of two messages are encrypted in a challenge oracle LR. The adversary is also equipped with oracles providing regular encryptions, rekey tokens, and re-encryptions. Intuitively, if no computationally efficient adversary can guess b then not even a single bit of information about messages leaks from ciphertexts, rekey tokens, and re-encryptions.

A tricky issue is how to rule out trivial wins, as allowing any combination of queries would lead to a vacuous definition — no correct updatable encryption scheme could meet it. We therefore disallow the adversary from receiving the information needed to rotate a challenge ciphertext (i.e., one returned from LR), or any ciphertext derived from a challenge ciphertext via rotations, to a compromised key. For example, the adversary should not be able to query $(\tilde{C}^*, C^*) \leftarrow \text{LR}(1, m_0, m_1)$, then $\text{ReKeyGen}(1, t+1, \tilde{C})$ and obtain the associated rekey token. If it could, then it would trivially be able to determine the challenge bit by applying the rekey token and then decrypting with the compromised key k_{t+1}.

To rule out such trivial wins,[2] the game therefore keeps track of the ciphertexts that derive from challenge ciphertexts, meaning those returned from LR. The table $\mathtt{T}$ performs this task. Its rows are indexed by key index, ciphertext header pairs, and entries are the header's associated ciphertext body. Tables in our pseudocode games are always initially everywhere set to $\perp$. Upon a rekey generation query, the game checks if the query is *invalid* — the target key is compromised ($j > t$) and $\mathtt{T}[i, \tilde{C}] \neq \perp$. If so, it returns $\perp$, rejecting the query as invalid. Otherwise, a rekey token is generated, and should $\mathtt{T}[i, \tilde{C}] \neq \perp$, then a new entry in $\mathtt{T}$ is generated by applying the deterministic re-encryption algorithm ReEnc to the ciphertext $(\tilde{C}, C)$ using the just-generated rekey token.

Upon a re-encryption query, a similar set of checks occurs, except that here invalid queries return the new ciphertext header (rather than the more obvious choice of the failure symbol $\perp$). Allowing the new ciphertext header to be returned for invalid re-encryption queries leads to a stronger definition, compared to the alternative of returning $\perp$. We believe there is merit to achieving this stronger notion, which requires schemes to somehow ensure that ciphertext headers alone are insufficient for decrypting message data. In particular, achieving it may lead to extra security robustness in the face of exposure of ciphertext headers, which are necessarily transmitted during rotation workflows, and so potentially more likely to be compromised by attackers.

We associate to an adversary $\mathcal{A}$, updatable AE scheme Π, numbers κ, t the advantage measure:

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,\kappa,t}(\mathcal{A}) = 2 \cdot \Pr\left[\text{UP-IND}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] - 1 \ .$$

We will informally say that a scheme is UP-IND-secure if no adversary using a "reasonable" amount of computation and queries can achieve advantage "far" from zero.

Ciphertext-independent schemes. Recall from the previous section that ciphertext-independent schemes do not have a ciphertext header ($\tilde{C} = \varepsilon$). For ciphertext-independent schemes, game UP-IND effectively treats $\tilde{C}$ as absent, meaning $\mathtt{T}$ entries are indexed simply by keys and $\tilde{C}$ is dropped from oracle interfaces and algorithms. But no ciphertext-independent scheme can meet UP-IND security. Consider the following UP-IND$_{\Pi,\kappa,t}$-adversary for Π a ciphertext-independent scheme, and numbers $\kappa > 0$ and $t > 0$. The adversary starts by querying ReKeyGen$(1, t+1)$ (where we have removed $\tilde{C}$ from notation) and because $\mathtt{T}$ has no entries, the oracle will return a valid rekey token $\Delta_{1,t+1}$. The adversary then queries LR$(1, m_0, m_1)$ for $m_0 \neq m_1$, and receives back a ciphertext C. It uses the rekey token to re-encrypt C to a new ciphertext C' decryptable under k_{t+1}, and uses it to determine which message was encrypted and, in turn, the challenge bit. This adversary achieves advantage 1.

This shows that the definition, which fundamentally relies on ciphertext headers, is too strong for any ciphertext-independent schemes. We therefore can give a weaker notion of confidentiality, which matches the notion from BLMR. We call this notion UP-IND-BI, and let the associated game UP-IND-BI be the same as UP-IND except that: (1) ReKeyGen immediately returns $\perp$ if $\mathsf{Hon}(i) \wedge \mathsf{Mal}(j)$ or $\mathsf{Hon}(j) \wedge \mathsf{Mal}(i)$, i.e., exactly one of the two keys is compromised; (2) ReEnc immediately returns $\perp$ if $\mathsf{Mal}(j)$, i.e., the target key is compromised; and (3) all references to $\mathtt{T}$ are removed. For a ciphertext-independent scheme Π and numbers κ, t, we associate to an UP-IND-BI$_{\Pi,\kappa,t}$-adversary $\mathcal{A}$ the advantage measure:

$$\mathsf{Adv}^{\text{up-ind-bi}}_{\Pi,\kappa,t}(\mathcal{A}) = 2 \cdot \Pr\left[\text{UP-IND-BI}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] - 1 \ .$$

A few observations are in order. First, the notion allows for not only ciphertext-independent but also bidirectional schemes, because of the strong restriction of disallowing attackers from obtaining rekey tokens $\Delta_{i,j,\tilde{C}}$ with $\mathsf{Mal}(i)$ (so the corresponding key is compromised), but with $\mathsf{Hon}(j)$ (an uncompromised key). Thus, in principle, bidirectional schemes could meet this notion, explaining our naming convention for the notion. Second, it is apparent that the BLMR invalidity procedures are, necessarily for bidirectional security, less permissive than ours, leading to a weaker security notion: the BLMR procedures are not ciphertext-specific but instead depend only on the compromise status of keys. Third, this security notion coincides with symmetric proxy re-encryption security (as also introduced in [BLMR15]).

Given that bidirectional, ciphertext-independent schemes have certain advantages in terms of performance and deployment simplicity, practitioners may prefer them in some cases. For that flexibility, one trades off control over the specificity of rekey tokens. This could be dangerous to confidentiality in some compromise scenarios.

[2] In the prior version of this paper, we used recursively defined invalidity procedures that (implicitly) operated on query transcripts to determine if a query was valid. We feel that the new pseudocode approach followed here is more precise.

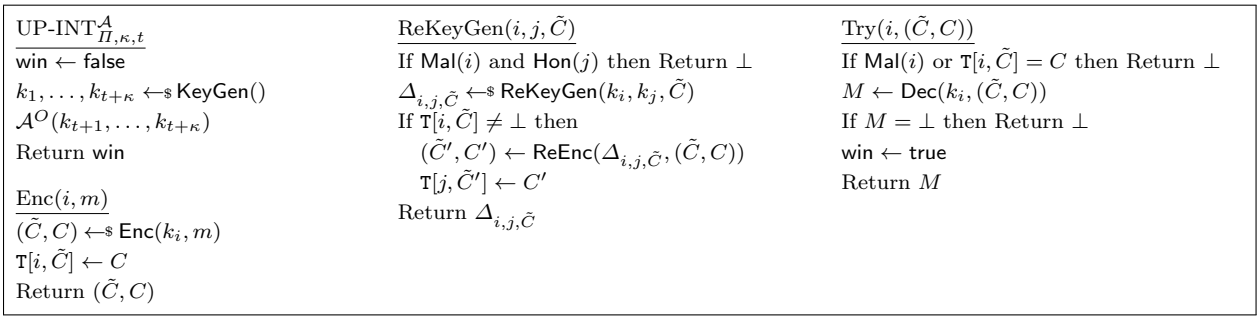

Fig. 4. Integrity security game for updatable AE schemes.

Future extensions. Our security notion for confidentiality (and, as we will see, for ciphertext integrity and re-encryption indistinguishability), only considers a static corruption of keys. We leave to future work an investigation of adaptive corruptions.

4.2 Ciphertext Integrity

We now turn to a notion of integrity. Informally, the adversary's goal is to generate a malicious ciphertext that nevertheless decrypts properly. Of course, we must exclude the adversary from simply resubmitting valid ciphertexts produced by an honest encryption oracle, or derived from such an encryption by way of re-encryption queries or rekey tokens. Hence we must track all ciphertexts which can be trivially derived from oracle queries, which is a challenging restriction to enforce in full generality because the game does not even see ciphertexts generated by a corrupted key. For example, a definition that allows an adversary to generate a rekey token from a compromised to an uncompromised key could simply rotate an adversarially generated, valid ciphertext to a valid one under an uncompromised key. We therefore take the approach of restricting the adversary from receiving re-keying tokens from a compromised key to an uncompromised key, since otherwise this would permit re-encrypting of unseen ciphertexts. Placing a similar restriction on re-encryption queries (of preventing rotations of a ciphertext under a compromised key to a ciphertext under a non-compromised key) would make the oracle not useful to the adversary, and so we omit having a re-encryption oracle entirely.

Formally, we capture this security via the game UP-INT shown in Figure 4. It is parameterized by an updatable AE scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Dec})$, two numbers $t > 0$ and $\kappa \geq 0$, and (implicitly) an adversary $\mathcal{A}$. As before t of the keys are honest, with predicates $\mathsf{Hon}(i) = (i \leq t)$ and $\mathsf{Mal}(i) = (i > t)$ indicating compromise status. To restrict the adversary from winning trivially, the Try oracle makes use of a table $\mathtt{T}$ that tracks ciphertexts under honest keys generated by the game or that are trivially known to the adversary via rekey tokens.[3] The latter relies on ReEnc being deterministic, a restriction we made earlier. We associate to an updatable encryption scheme Π, numbers $\kappa \geq 0$ and $t > 0$, and (implicitly) an adversary $\mathcal{A}$, the advantage measure:

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,\kappa,t}(\mathcal{A}) = \Pr\left[\text{UP-INT}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] .$$

A few notes about our definition are in order. The restrictions on the adversary means that our security model does *not* cover attacks in which an adversary is able to inject new ciphertexts for re-encryption during a key compromise. That is, the adversary inserts into a database maliciously generated ciphertexts (under a compromised key) that then get rotated to a new key. Such attacks seem impossible to avoid. That said, our definition also allows attacks that would seem to be avoidable at least in principal. For example, consider an attack in which an adversary rotates a malicious ciphertext to an honest key, and then later manipulates the new ciphertext. This is allowed by our definition, but would not seem to work against the trivial non-compact scheme that sets $\tilde{C}$ to a (standard) AE of the message and rotates by decrypting and re-encrypting. We leave to future work an exploration of stronger notions, noting only that in considering this aim, we have had trouble in finding ways of meaningfully defining invalid queries for more permissive definitions.

[3] In the prior version of this paper, we used a recursively defined invalidity procedure that (implicitly) operated on query transcripts to determine if a query was valid. The CTXT invalidity procedure used in the previous version actually resulted in a vacuous UP-INT security goal (no scheme could achieve it).

$\underline{\mathsf{Enc}(k, m)}$	$\underline{\mathsf{ReKeyGen}(k_1, k_2, \tilde{C})}$	$\underline{\mathsf{ReEnc}(\Delta_{1,2,\tilde{C}}, (\tilde{C}, C))}$	$\underline{\mathsf{Dec}(k, (\tilde{C}, C))}$
$x \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{dem}}$	$x \leftarrow \mathcal{D}_{\mathrm{kem}}(k_1, \tilde{C})$	Return $(\Delta_{1,2,\tilde{C}}, C)$	$x \leftarrow \mathcal{D}_{\mathrm{kem}}(k, \tilde{C})$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k, x)$	If $x = \perp$ then Return $\perp$		If $x = \perp$ then Return $\perp$
$C \leftarrow \mathcal{E}_{\mathrm{dem}}(x, m)$	$\Delta_{1,2,\tilde{C}} \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k_2, x)$		$m \leftarrow \mathcal{D}_{\mathrm{dem}}(x, C)$
return $(\tilde{C}, C)$	Return $\Delta_{1,2,\tilde{C}}$		Return m

Fig. 5. Algorithms for the AE-hybrid updatable AE scheme for AE schemes $\pi_{\mathrm{kem}} = (\mathcal{K}_{\mathrm{kem}}, \mathcal{E}_{\mathrm{kem}}, \mathcal{D}_{\mathrm{dem}})$ and $\pi_{\mathrm{dem}} = (\mathcal{K}_{\mathrm{dem}}, \mathcal{E}_{\mathrm{dem}}, \mathcal{D}_{\mathrm{dem}})$; KeyGen for AE-Hybrid simply runs $\mathcal{K}_{\mathrm{kem}}$.

5 The Basic KEM/DEM Construction

We first investigate the security of updatable AE schemes built using the KEM/DEM approach sketched in the introduction. Such schemes are currently in widespread use, for example in AWS's and Google's cloud storage systems [AWS,Goo]. But to the best of our knowledge they have received no formal analysis to date. Using the confidentiality and integrity definitions from the previous section, we discover that this construction does not achieve confidentiality or integrity in our models when the adversary can compromise at least one key. Confidentiality can't be achieved because our security notion returns ciphertext headers in response to invalid re-encryption queries. We do show security in our models when the number of compromised keys κ is equal to 0.

Figure 5 defines an updatable AE scheme, AE-hybrid, that uses an AE scheme $\pi_{\mathrm{kem}} = (\mathcal{K}_{\mathrm{kem}}, \mathcal{E}_{\mathrm{kem}}, \mathcal{D}_{\mathrm{dem}})$, called the key encapsulation mechanism (KEM), and an AE scheme $\pi_{\mathrm{dem}} = (\mathcal{K}_{\mathrm{dem}}, \mathcal{E}_{\mathrm{dem}}, \mathcal{D}_{\mathrm{dem}})$, called the data encapsulation mechanism (DEM).[4] Encryption is a standard hybrid encryption, and key rotation is straightforward: decrypting the DEM key and re-encrypting it under the target key. Using this scheme means that re-keying requires constant time and communication, independent of the length of the plaintext. In fact, we note that this scheme (or slight variants of it) is seeing widening deployment for encrypted cloud storage services. Both Amazon Web Services [AWS] and Google Cloud Platform [Goo] use AE-hybrid to perform key rotations over encrypted customer data.

Message confidentiality of AE-hybrid. We first demonstrate that AE-hybrid is not UP-IND secure when key compromises are allowed (i.e., $\kappa > 0$). The reason is that in our games we return the header for invalid ReEnc queries. Letting Π be the AE-hybrid scheme, consider the following $\text{UP-IND}_{\Pi,1,1}$ adversary $\mathcal{A}$. It makes an initial query to $\mathrm{LR}(1, m_0, m_1)$ for distinct messages $m_0 \neq m_1$ and receives challenge ciphertext $(\tilde{C}^*, C^*) = (\mathcal{E}_{\mathrm{kem}}(k_1, x), \mathcal{E}_{\mathrm{dem}}(x, m_b))$. The adversary subsequently calls $\mathrm{ReEnc}(1, 2, (\tilde{C}^*, C^*))$. Because k_2 is corrupted, the adversary receives just the re-encrypted ciphertext header $\tilde{C}' = \mathcal{E}_{\mathrm{kem}}(k_2, x)$. The adversary sets $x = \mathcal{D}_{\mathrm{kem}}(k_2, \tilde{C}')$ using its knowledge of k_2, and computes $m_b = \mathcal{D}_{\mathrm{dem}}(x, C^*)$. It checks whether $m_b = m_0$ or $m_b = m_1$, and guesses the bit b appropriately.

We can change the security notion to return $\perp$ upon re-encryption queries to a corrupted key, and this attack fails. We omit a formal analysis of security in this case. Instead we analyze security when no keys are compromised (i.e., $\kappa = 0$), as captured in the following theorem.

Theorem 1 (UP-IND **security of AE-hybrid with no key compromise).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ and $\pi_{dem} = (\mathcal{K}_{dem}, \mathcal{E}_{dem}, \mathcal{D}_{dem})$ be AE schemes and Π be the updatable AE scheme AE-hybrid using π as defined in Figure 5. Let $t \geq 1$. Then for any* $\text{UP-IND}_{\Pi,0,t}$ *adversary $\mathcal{A}$ making at most q queries, we give adversaries $\mathcal{B}, \mathcal{C}$ such that:*

$$\mathsf{Adv}^{\text{up-ind}}_{\Pi,0,t}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{dem}}(\mathcal{C}).$$

Adversaries $\mathcal{B}, \mathcal{C}$ run in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q + t)$ overhead and make at most q queries.

The proof is relatively straightforward: apply the security of the KEM and then one can apply the one-time, real-or-random security of the DEM. This makes the returned transcript simply random values, independent of the challenge bit b. It is important to note that we need the KEM to resist chosen-ciphertext attacks here because the adversary gets to submit KEM ciphertexts to ReKeyGen and ReEnc.

[4] We are slightly abusing terminology, since KEMs typically do not allow inputs and aren't, strictly speaking, AE schemes. Here we could use KEMs that do not take message inputs, but in subsequent constructions we will need true AE schemes that take inputs.

Ciphertext integrity of AE-hybrid. The AE-hybrid scheme is insecure in the UP-INT sense when $\kappa \geq 1$. The primary reason is that ciphertext headers and bodies are not cryptographically bound to each other, allowing an adversary to mix and match headers with bodies by means of decrypting a rekey token.

Letting Π be the AE-hybrid scheme, consider the following UP-INT$_{\Pi,1,1}$ adversary $\mathcal{A}$. First, the adversary queries Enc$(1, m)$ to obtain an encryption $(\tilde{C}, C) = (\mathcal{E}_{\text{kem}}(k_1, x), \mathcal{E}_{\text{dem}}(x, m))$, and subsequently queries ReKeyGen$(1, 2, \tilde{C})$, receiving the rekey token $\tilde{C}' = \mathcal{E}_{\text{kem}}(k_2, x)$. Since $\mathcal{A}$ has key k_2, it recovers x by decrypting $\tilde{C}'$. Finally, $\mathcal{A}$ queries Try$(1, (\tilde{C}, \mathcal{E}_{\text{dem}}(x, m')))$ for some $m' \neq m$. This will successfully set win.

As with message confidentiality, we can, however, prove UP-INT security of AE-hybrid when $\kappa = 0$.

Theorem 2 (UP-INT **Security of AE-hybrid with no key compromise).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ and $\pi_{dem} = (\mathcal{K}_{dem}, \mathcal{E}_{dem}, \mathcal{D}_{dem})$ be AE schemes and Π be the updatable AE scheme AE-hybrid using π as defined in Figure 5. Let $t \geq 1$. Then for any* UP-INT$_{\Pi,0,t}$ *adversary $\mathcal{A}$ making at most q queries, we give adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathsf{Adv}^{\text{up-int}}_{\Pi,0,t}(\mathcal{A}) \leq \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem},t}(\mathcal{B}) + \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{dem}}(\mathcal{C}) \ .$$

Adversaries $\mathcal{B}, \mathcal{C}$ run in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q + t)$ overhead and make at most q queries.

To sketch the proof, we first apply the multi-use, AE security of the KEM. This ensures that the only valid KEM ciphertexts are those generated during queries to Enc or ReKeyGen, meaning the only DEM keys that can be used during a Try query are those chosen in Enc. We then apply the multi-use, AE security of the DEM scheme, making it so that the only valid DEM ciphertexts are those generated by Enc. Finally we must rule out mix-and-match attacks that combine one valid KEM ciphertext with a valid DEM ciphertext, but for a different DEM key than the one encapsulated in the KEM ciphertext. But such attacks don't work because (1) each DEM key at this point is associated with a distinct set of table entries (our multi-use AE security notion rules out DEM key collisions) and (2) the table $\mathtt{T}$ tracks all combinations of KEM and DEM ciphertexts generated during Enc and ReKeyGen queries.

6 The KEM/DEM with Secret Sharing (KSS) Scheme

We now introduce a scheme called KSS (for KEM/DEM with Secret Sharing). It modifies the previous KEM/DEM scheme in order to satisfy our confidentiality and ciphertext integrity notions, even when keys can be compromised. Recall that there were two issues that prevented the basic KEM/DEM scheme from achieving our security definitions. For confidentiality the problem was that revealing the rekey token for an honestly-generated ciphertext to a compromised key was sufficient to reveal the DEM key (because we reveal the ciphertext header on ReEnc queries). KSS deals with this by secret-sharing DEM keys across the KEM plaintext and the ciphertext body. Thus headers are no longer sufficient to recover the DEM key. For ciphertext integrity the problem was that the KEM and DEM ciphertexts were not cryptographically bound. KSS handles this by including the DEM ciphertext's tag as part of the KEM plaintext.[5]

More formally, let $\pi_{\text{kem}} = (\mathcal{K}_{\text{kem}}, \mathcal{E}_{\text{kem}}, \mathcal{D}_{\text{kem}})$ and $\pi_{\text{dem}} = (\mathcal{K}_{\text{dem}}, \mathcal{E}_{\text{dem}}, \mathcal{D}_{\text{dem}})$ be AE schemes. Then we use them to build the updatable encryption scheme KSS (KEM/DEM with Secret Sharing) as detailed in Figure 6. In terms of performance, KSS is competitive with the basic KEM/DEM scheme, AE-hybrid. It requires an additional $|y|$ bits for ciphertexts due to the DEM key sharing component y, but this can be as small as 128 bits. In terms of computation, KSS only adds a few basic operations on top of AE-hybrid.

Message confidentiality of KSS. We start by analyzing the UP-IND security of KSS, as captured by the following theorem.

Theorem 3 (UP-IND **Security of** KSS**).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ and $\pi_{dem} = (\mathcal{K}_{dem}, \mathcal{E}_{dem}, \mathcal{D}_{dem})$. be AE schemes, and let* KSS *be the updatable AE scheme built from them as defined in Figure 6. Let $t > 0$ and $\kappa \geq 0$. Then for any* UP-IND$_{\text{KSS},\kappa,t}$ *adversary $\mathcal{A}$ that makes at most q queries, we give adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathsf{Adv}^{\text{up-ind}}_{\text{KSS},\kappa,t}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{dem}}(\mathcal{C}) \ .$$

[5] One might wonder why we don't use as KEM an authenticated encryption with associated data (AEAD) scheme, and bind the DEM ciphertext to the AE by using the DEM's authentication tag as associated data for the KEM. This doesn't work because during rekey generation we need the tag for use with the new KEM ciphertext.

$\underline{\mathsf{Enc}(k, m)}$	$\underline{\mathsf{ReKeyGen}(k_i, k_j, \tilde{C})}$	$\underline{\mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, (r, C)))}$	$\underline{\mathsf{Dec}(k, (\tilde{C}, (r, C)))}$
$x \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{dem}}$	$(y \parallel \tau) \leftarrow \mathcal{D}_{\mathrm{kem}}(k_i, \tilde{C})$	$(\tilde{C}', r') \leftarrow \Delta_{i,j,\tilde{C}}$	$(y \parallel \tau) \leftarrow \mathcal{D}_{\mathrm{kem}}(k, \tilde{C})$
$r \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{dem}}$	If $(y \parallel \tau) = \perp$ then	Return $(\tilde{C}', (r \oplus r', C))$	If $(y \parallel \tau) = \perp$ then
$(C, \tau) \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{dem}}(x, m)$	Return $\perp$		Return $\perp$
$y \leftarrow x \oplus r$	$r' \leftarrow\!\!\$\ \mathcal{K}$		$x \leftarrow y \oplus r$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k, y \parallel \tau)$	$\tilde{C}' \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k_j, y \oplus r' \parallel \tau)$		$m \leftarrow \mathcal{D}_{\mathrm{dem}}(x, C, \tau)$
Return $(\tilde{C}, (r, C))$	$\Delta_{i,j,\tilde{C}} \leftarrow (\tilde{C}', r')$		Return m
	Return $\Delta_{i,j,\tilde{C}}$		

Fig. 6. The KSS updatable encryption scheme using AE schemes $\pi_{\mathrm{kem}} = (\mathcal{K}_{\mathrm{kem}}, \mathcal{E}_{\mathrm{kem}}, \mathcal{D}_{\mathrm{kem}})$ and $\pi_{\mathrm{dem}} = (\mathcal{K}_{\mathrm{dem}}, \mathcal{E}_{\mathrm{dem}}, \mathcal{D}_{\mathrm{dem}})$; KeyGen for KSS simply runs $\mathcal{K}_{\mathrm{kem}}$

Adversaries $\mathcal{B}, \mathcal{C}$ each make at most q queries and run in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q + t + \kappa)$ overhead.

Proof. Our proof uses a sequence of games. Game G_0 (Figure 7, boxed statements omitted) is the $\text{UP-IND}_{\mathrm{KSS},\kappa,t}$ game. In this game, and in all subsequent ones, we omit the main procedure to save space. That procedure picks a random bit b, generates secret keys $k_1, \ldots, k_{t+\kappa} \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{kem}}$, runs $\mathcal{A}$ with input $k_{t+1}, \ldots, k_{t+\kappa}$ and with access to the oracles, and returns true if $\mathcal{A}$'s output b' is equal to b.

Game G_1 (Figure 7, boxed statements included) replaces the KEM encryption when used with honest keys with randomly chosen ciphertexts and table look-ups to implement decryption. To bound the transition between G_0 and G_1 we use a reduction to the AE security of the KEM. In detail, let $\mathcal{B}$ be the $\text{MU-ROR-AE}_{\pi_{\mathrm{kem}},t}$ adversary that works exactly like game G_0 (and G_1) except that: (1) uses of $\mathcal{E}_{\mathrm{kem}}(k_i, m)$ with $\mathsf{Hon}(i)$ are replaced with a query to $\mathcal{B}$'s encryption oracle on (i, m) and (2) uses of $\mathcal{D}_{\mathrm{kem}}(k_i, \tilde{C})$ with $\mathsf{Hon}(i)$ are replaced with a query to $\mathcal{B}$'s decryption oracle on $(i, \tilde{C})$. By construction, it holds that $\text{MU-ROR-AE0}^{\mathcal{B}}_{\pi,t} = G_0^{\mathcal{A}}$ and $\text{MU-ROR-AE1}^{\mathcal{B}}_{\pi,t} = G_1^{\mathcal{A}}$. Thus,

$$\Pr[\, G_0 \Rightarrow \mathsf{true} \,] \leq \Pr[\, G_1 \Rightarrow \mathsf{true} \,] + \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi,t}(\mathcal{B}) \; .$$

We now will make transitions so that the transcript observed by the adversary is independent of the challenge bit b. This will involve: (1) showing that headers returned in response to invalid ReEnc queries are indistinguishable from encryptions of a random DEM key and the tag (because of the secret-sharing of DEM keys and refreshing of them during re-encryption); (2) the y values generated during challenge queries are never revealed to the adversary (which we argue by induction using the rules preventing certain rekey or re-encryption queries); and (3) applying the OT-ROR security of π_{dem} to argue that the ciphertext body leaks nothing about the challenge bit b.

Towards this we start by modifying how ReEnc returns headers in the case of an invalid query (one for which $\mathtt{T}[i, \tilde{C}] \neq \perp$ and $\mathsf{Mal}(j)$). Notice that in game G_1, in the case that just the header is returned in response to a ReEnc query, the header's plaintext portion $y \oplus r'$ is independent of y because r' is uniform and not used elsewhere in the game. We make this explicit in game G_2 (Figure 7, boxed statement omitted). Game G_2 also dispenses with the now redundant lines of code from G_1's handling of $\mathrm{E}_{\mathrm{kem}}$, $\mathrm{D}_{\mathrm{kem}}$. We have that $\Pr[G_1^{\mathcal{A}} \Rightarrow \mathsf{true}] = \Pr[G_2^{\mathcal{A}} \Rightarrow \mathsf{true}]$. Note that this step of the proof relies on the secret sharing of the DEM key — without it we would not be able to argue independence and, in fact, the returned ciphertext header would reveal the DEM key.

In game G_3, we replace the DEM key share y in LR with a constant value. We will argue that

$$\Pr\left[\, G_2^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] = \Pr\left[\, G_3^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] , \tag{1}$$

by arguing that no values returned to the adversary depend on y. Intuitively, the reason is that while y may affect other entries of $\mathtt{C}$, via rekey or re-encryption queries, the rules about invalid queries combined with our earlier game transition to G_2 ensure that these plaintext values never impact any values returned to the adversary.

To argue this, consider any query to LR on index i^* that produces an entry $\mathtt{C}[i^*, \tilde{C}^*] = y \| \tau$. Necessarily $\mathtt{T}[i^*, \tilde{C}^*] \neq \perp$. We perform an inductive argument over the subsequent queries to show the following condition holds after each query: the only values in the game dependent on y are entries in $\mathtt{C}[j, \tilde{C}]$ for some $j, \tilde{C}$ for which $\mathtt{T}[j, \tilde{C}] \neq \perp$. In particular, no values returned to the adversary are dependent. By dependent we mean

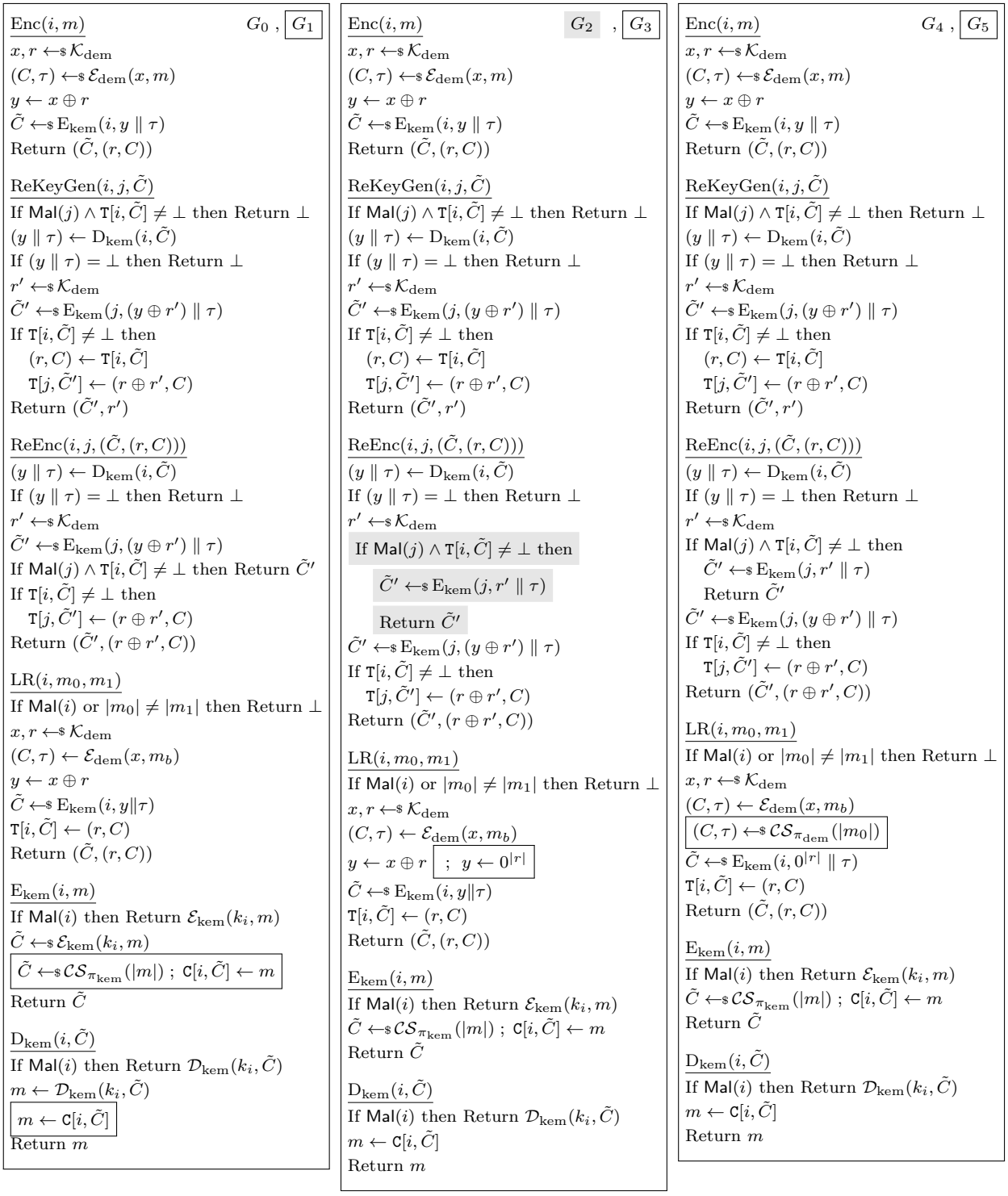

Fig. 7. Games for proof of UP-IND for KSS.

that the values are a function of y and (possibly) other variables defined in the game. Below we refer to entries $j, \tilde{C}$ for which $\mathtt{C}[j, \tilde{C}]$ depends on y as dependent entries.

As base case consider that the LR query was the last query. Then the condition holds by inspection of the code of LR in game G_2. Now consider an arbitrary later query, and assume that the condition holds at the end of the previous query. We have the following cases broken down by the type of query.

(1) The query is to Enc. Then no dependent entries are accessed, and so the condition trivially holds after the query.

(2) The query is to ReKeyGen on a triple $i, j, \tilde{C}$. If $i, \tilde{C}$ is not a dependent entry, then the condition trivially holds after the query. Otherwise, by the inductive hypothesis $\mathtt{T}[i, \tilde{C}] \neq \bot$ and so for $\mathtt{C}[i, \tilde{C}]$ to be accessed, necessarily $\mathsf{Hon}(j)$. Then the procedure generates a new dependent plaintext submitted to $\mathrm{E}_{\mathrm{kem}}$ for $\mathsf{Hon}(j)$, thereby adding a new dependent entry $\mathtt{C}[j, \tilde{C}']$. Because $\mathtt{T}[i, \tilde{C}] \neq \bot$ it holds that $\mathtt{T}[j, \tilde{C}']$ is set to a value other than $\bot$. None of the returned values depend on either dependent entry. Thus the condition holds after the query completes.

(3) The query is to ReEnc on a triple $i, j, (\tilde{C}, (r, C))$. If $i, \tilde{C}$ is not a dependent entry, then the condition trivially holds after the query. Otherwise, by the inductive hypothesis $\mathtt{T}[i, \tilde{C}] \neq \bot$. If $\mathsf{Mal}(j)$ then no new dependent entries are generated (the KEM plaintext in this case using r' instead of $y \oplus r'$). If $\mathsf{Hon}(j)$, then the procedure generates a new dependent plaintext submitted to $\mathrm{E}_{\mathrm{kem}}$ for $\mathsf{Hon}(j)$, adding a new dependent entry $\mathtt{C}[j, \tilde{C}']$. Because $\mathtt{T}[i, \tilde{C}] \neq \bot$ it holds that $\mathtt{T}[j, \tilde{C}']$ is set to a value other than $\bot$. None of the returned values depend on either dependent entry. Thus the condition holds after the query completes.

Thus by induction the condition holds for all queries, and we have justified (1).

Game G_4 (Figure 7, boxed statement omitted) is the same as G_3 except that the extraneous y computations in LR are removed. Game G_5 (Figure 7, boxed statement included) replaces C, τ in LR with a random ciphertext. We justify the transition from G_4 to G_5 via a reduction to the one-time, real-or-random security of π_{dem}. In more detail, let $\mathcal{C}$ be the OT-ROR$_{\pi_{\mathrm{dem}}}$ adversary that works as follows. It runs game G_4 except that in queries it replaces computing the DEM encryption in LR by a query to its oracle on (c, m_b) for a counter value c that it increments with each execution of LR (the counter ensuring distinct keys are used each time). Adversary $\mathcal{C}$ outputs one if $(b = b')$ after $\mathcal{A}$ finishes. We have by construction that $\Pr[\text{MU-ROR1}^{\mathcal{C}}_{\pi_{\mathrm{dem}}} \Rightarrow 1] = \Pr[G_4^{\mathcal{A}} \Rightarrow \mathsf{true}]$. We also have that $\Pr[\text{MU-ROR0}^{\mathcal{C}}_{\pi_{\mathrm{dem}}} \Rightarrow 1] = \Pr[G_5^{\mathcal{A}} \Rightarrow \mathsf{true}]$. Thus,

$$\Pr\left[\, G_4^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] = \Pr\left[\, G_5^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{dem}}}(\mathcal{C}) \ .$$

Note that the adversary $\mathcal{C}$ makes at most q queries in total and at most one query on any key index i.

In game G_5 the bit b is not used (once we remove the extraneous code above the boxed statement). Thus $\Pr[G_5^{\mathcal{A}} \Rightarrow \mathsf{true}] = 1/2$. Combining all the above we have the following:

$$\begin{aligned}
\mathsf{Adv}^{\text{up-ind}}_{\mathrm{KSS},\kappa,t}(\mathcal{A}) &= 2 \cdot \Pr\left[\, \text{UP-IND}^{\mathcal{A}}_{\mathrm{KSS},\kappa,t} \Rightarrow \mathsf{true} \,\right] - 1 \\
&= 2 \cdot \Pr\left[\, G_0^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] - 1 \\
&\leq 2 \cdot \Pr\left[\, G_1^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\, G_2^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\, G_3^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\, G_4^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&\leq 2 \cdot \Pr\left[\, G_5^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{dem}}}(\mathcal{C}) - 1 \\
&= 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{dem}}}(\mathcal{C}) \ .
\end{aligned}$$

□

Ciphertext integrity of KSS. We now turn to analyzing the UP-INT security of KSS. The new scheme protects against ciphertext tampering in two ways. First, the KEM is itself an AE scheme. Second, we include the DEM authenticity tag τ within the KEM's plaintext. The latter should, intuitively, ensure that any tampering with the ciphertext body, which modifies the decrypted message since ciphertext body and

message bits are 1-1, are caught. However the complexities introduced by rekey tokens means there are some subtleties that this brief intuition omits.

Most notably, our UP-INT security notion allows the adversary to easily obtain the DEM key associated with an honest ciphertext. To see this, consider the adversary that queries $(\tilde{C}, (r, C)) \leftarrow\!\!\$\ \mathrm{Enc}(1, m)$ for an arbitrary message m; then queries $\Delta_{1,t+1,\tilde{C}} \leftarrow\!\!\$\ \mathrm{ReKeyGen}(1, t+1, \tilde{C})$ which equals $(\tilde{C}', r')$; computes $(\tilde{C}', (r \oplus r', C')) \leftarrow \mathsf{ReEnc}(\Delta_{1,t+1,\tilde{C}}, (\tilde{C}, (r, C)))$; decrypts the resulting ciphertext to obtain y; and finally lets $x \leftarrow y \oplus x'$. That our definition permits such sequences of queries means we cannot rely on standard properties that require x to be secret.

We therefore require that our DEM be *compactly robust*. This is a new notion, closely related to robustness [ABN10,FLPQ13,FOR17] and being compactly committing [GLR17]. In words, no adversary should be able to find two secret keys and two ciphertexts that have the same authentication tag such that the ciphertexts decrypt to some messages under the two keys. More formally, let π be an AE scheme. Then we define the game ROB_π as follows. An adversary $\mathcal{A}$ is run, and outputs a tag τ, two keys x_1, x_2, and two ciphertext portions C_1, C_2. Then the game outputs true if: (1) $(x_1, C_1) \neq (x_2, C_2)$; (2) $\mathcal{D}(x_1, (C_1, \tau)) \neq \perp$; and (3) $\mathcal{D}(x_2, (C_2, \tau)) \neq \perp$. We associate to π and a ROB_π adversary $\mathcal{A}$ the advantage measure

$$\mathsf{Adv}^{\mathrm{rob}}_{\pi}(\mathcal{A}) = \Pr\left[\,\mathrm{ROB}^{\mathcal{A}}_{\pi} \Rightarrow \mathsf{true}\,\right] .$$

This notion can be seen as a strengthening of the full robustness (FROB) property of [FOR17] to only require that the tag, and not the entire ciphertext, be the same for the two keys. It can also be seen as a strengthening of the strong receiver binding notion of [DGRW18], somewhat analogous to the strengthening from plaintext integrity to ciphertext integrity. Not all AE schemes provide this level of robustness, but some standard ones do. For example, single-key Encrypt-then-MAC with AES-CTR mode and HMAC does so. One can adapt the proof of [GLR17, Th. 3] to show that this construction is compactly robust.

Theorem 4 (UP-INT **Security of** KSS**).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ and $\pi_{dem} = (\mathcal{K}_{dem}, \mathcal{E}_{dem}, \mathcal{D}_{dem})$. be AE schemes, and let* KSS *be the updatable AE scheme built from them as defined in Figure 6. Let $t > 0$ and $\kappa \geq 0$. Then for any* UP-INT$_{\Pi,\kappa,t}$*-adversary $\mathcal{A}$ making at most q queries we give adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathsf{Adv}^{\text{up-int}}_{\mathrm{KSS},\kappa,t}(\mathcal{A}) \leq \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem},t}(\mathcal{B}) + \mathsf{Adv}^{\mathrm{rob}}_{\pi_{dem}}(\mathcal{C}) .$$

Adversaries $\mathcal{B}, \mathcal{C}$ each run in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q + t + \kappa)$ overhead. Adversary $\mathcal{B}$ makes at most q queries.

Proof. Our proof uses a sequence of two games. Game G_0 (Figure 8) implements $\text{UP-INT}^{\mathcal{A}}_{\mathrm{KSS},\kappa,t}$. In both games we omit the main procedure. That procedure generates secret keys $k_1, \ldots, k_{t+\kappa} \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{kem}}$, runs $\mathcal{A}$ with input $k_{t+1}, \ldots, k_{t+\kappa}$ and with access to the oracles, and returns true if $\mathsf{win} = \mathsf{true}$ after $\mathcal{A}$ finishes executing.

Game G_1 (Figure 8, boxed statements included) replaces the KEM encryption when used with honest keys with randomly chosen ciphertexts and table look-ups to implement decryption. To bound the transition between G_0 and G_1 we use a reduction to the AE security of the KEM that is almost identical to the same reduction in the proof of Theorem 3. In detail, let $\mathcal{B}$ be the $\text{MU-ROR-AE}_{\pi_{\mathrm{kem}},t}$ adversary that works exactly like game G_0 (and G_1) except that: (1) uses of $\mathcal{E}_{\mathrm{kem}}(k_i, m)$ with $\mathsf{Hon}(i)$ are replaced with a query to $\mathcal{B}$'s encryption oracle on (i, m) and (2) uses of $\mathcal{D}_{\mathrm{kem}}(k_i, \tilde{C})$ with $\mathsf{Hon}(i)$ are replaced with a query to $\mathcal{B}$'s decryption oracle on $(i, \tilde{C})$. By construction, it holds that $\text{MU-ROR-AE0}^{\mathcal{B}}_{\pi_{\mathrm{kem}},t} = G^{\mathcal{A}}_0$ and $\text{MU-ROR-AE1}^{\mathcal{B}}_{\pi_{\mathrm{kem}},t} = G^{\mathcal{A}}_1$. Thus,

$$\Pr\left[\,G_0 \Rightarrow \mathsf{true}\,\right] \leq \Pr\left[\,G_1 \Rightarrow \mathsf{true}\,\right] + \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) .$$

Let us refer to the first query to Try that sets win to true as a winning query. It is without loss to assume that no query follows a winning query, making it the winning query. Consider the winning query and let its input be $i^*, (\tilde{C}^*, (r^*, C^*))$. We will show that there exists another query that, together with the winning query, defines values $\tau^*, (x, C), (x^*, C^*)$ that suffice for winning game $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$. We will therefore be able to define a reduction implying that $\mathcal{A}$ winning in game G_1 implies violating the compact robustness of π_{dem}.

Towards this, note that in game G_1 it must be that the winning query accesses $\mathtt{C}[i^*, \tilde{C}^*]$ because i^* must be an honest key. Let $x^*, y^*, r^*, C^*, m^*, \tau^*$ be the variables in Try associated to the winning query, with $\mathtt{C}[i^*, \tilde{C}^*] = (r^*, C^*)$. We have that

$$\mathcal{D}_{\mathrm{dem}}(x^*, (C^*, \tau^*)) \neq \perp \tag{2}$$

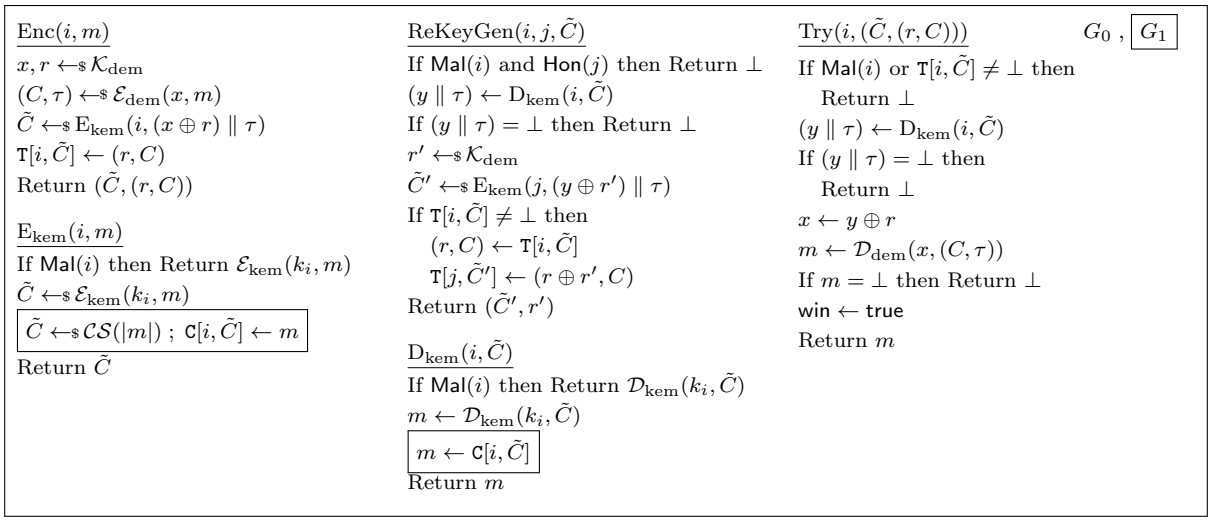

Fig. 8. Games for proof of UP-INT for KSS.

because it was a winning query. Since $\mathtt{C}[i^*, \tilde{C}^*] \neq \bot$, there exists a previous query $\mathrm{Enc}(i^*, m)$ or $\mathrm{ReKeyGen}(i, i^*, \tilde{C})$ that set $\mathtt{C}[i^*, \tilde{C}^*]$.

First consider the former case, that the previous query was to $\mathrm{Enc}(i^*, m)$. Necessarily this generated a valid ciphertext (C, τ) and random value r such that $\mathtt{T}[i^*, \tilde{C}^*] = (r, C)$. It must be that $\tau = \tau^*$ and $x \oplus r = y^*$ because $\mathtt{C}[i^*, \tilde{C}^*] = (y^* \| \tau^*)$. Moreover, because $\mathtt{T}[i^*, \tilde{C}^*] \neq (r^*, C^*)$ for a winning query, it must be that $(r, C) \neq (r^*, C^*)$. Thus either $x \neq x^*$ or $C \neq C^*$. We therefore let our $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$ winning values be $\tau^*, (x, C), (x^*, C^*)$.

Second consider the other case, that the previous query was to $\mathrm{ReKeyGen}(i, i^*, \tilde{C})$. Assume for the moment that this query set $\mathtt{T}[i^*, \tilde{C}^*] = (r \oplus r', C)$. Let r, r', y, C, τ be the values defined in handling the rekey generation query. For the Try query to be winning, then, it must be that $(r \oplus r', C) \neq (r^*, C^*)$. Because this rekey query set $\mathtt{C}[i^*, \tilde{C}^*] = y^* \| \tau^* = y \| \tau$ we have that $\tau = \tau^*$ and that:

$$x \oplus (r \oplus r') = y = y^* = x^* \oplus r^*$$

where we let $x = y \oplus (r \oplus r')$. Combining the above we have that $(x, C) \neq (x^*, C^*)$ and so we can set the $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$ winning values to be $\tau^*, (x, C), (x^*, C^*)$.

What remains is to argue that $\mathtt{T}[i^*, \tilde{C}^*]$ was indeed necessarily set in this query. We do so by proving that for $\mathrm{ReKeyGen}(i, i^*, \tilde{C})$ to set $\mathtt{C}[i^*, \tilde{C}^*]$ there must have previously been a query $(\tilde{C}_0, (r_0, C_0)) \leftarrow\!\!\$\ \mathrm{Enc}(i_0, m)$ followed by $\gamma \geq 1$ queries:

$$(\tilde{C}_1, x_1, r_1) \leftarrow\!\!\$\ \mathrm{ReKeyGen}(i_0, i_1, \tilde{C}_0)\ ,\ \ldots\ ,\ (\tilde{C}^*, x', r') \leftarrow\!\!\$\ \mathrm{ReKeyGen}(i_{\gamma-1}, i^*, \tilde{C}_{\gamma-1})\ .$$

where we've let $i_{\gamma-1} = i$ and $\tilde{C}_{\gamma-1} = \tilde{C}$. The last query represents the rekey generation query that set $\mathtt{C}[i^*, \tilde{C}^*]$. For the latter to have occurred, it must be that key $i_{\gamma-1}$ is honest otherwise the query would have immediately returned $\bot$ (because i^* is also honest). A previous query must therefore have set $\mathtt{C}[i_{\gamma-1}, \tilde{C}_{\gamma-1}]$. If that query was an ReKeyGen query, then repeat the above reasoning until one arrives at the $(\tilde{C}_0, (r_0, C_0)) \leftarrow\!\!\$\ \mathrm{Enc}(i_0, \tilde{C}_0)$ query. Now in this query it is the case that $\mathtt{T}[i_0, \tilde{C}_0]$ was necessarily set to a value other than $\bot$. In turn, the next $\mathrm{ReKeyGen}(i_0, i_1, \tilde{C}_0)$ query had $\mathtt{T}[i_0, \tilde{C}_0] \neq \bot$, and therefore set $\mathtt{T}[i_1, \tilde{C}_1]$ to a value, and so on until we have that $\mathtt{T}[i^*, \tilde{C}^*]$ is set to something other than $\bot$.

In all cases we have shown that a winning query leads to a $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$ winning set of values. Let $\mathcal{C}$ be the $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$-adversary that runs G_1 and, upon a winning query, determines the previous ReKeyGen query and generates the appropriate winning pair. This can be done in time sub-linear in the number of queries by searching over entries in $\mathtt{C}$. By our arguments above, we have that:

$$\Pr\left[\, G_1^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] \leq \mathsf{Adv}^{\mathrm{rob}}_{\pi_{\mathrm{dem}}}(\mathcal{C})\ .$$

□

7 Indistinguishability of Re-encryptions

The KSS scheme in the previous section achieves message confidentiality and ciphertext integrity, even though the actual DEM key is not modified in the course of performing a rotation. Modifying the scheme to ensure the DEM key is also rotated is non-trivial, requiring either significant communication complexity (linear in the length of the encrypted message) between the key server and storage, or the introduction of more advanced primitives such as key-homomorphic PRFs. The question that arises is whether or not changing DEM keys leaves KSS vulnerable to attacks not captured by the definitions introduced thus far.

BLMR's brief treatment of updatable encryption addresses this issue by requiring that all randomness be refreshed during a rotation. Intuitively this would seem to improve security, but the goal they formalize for this, detailed below, is effectively a correctness condition (i.e., it does not seem to account for adversarial behaviors). It doesn't provide much intuition about what attacks would be ruled out by changing DEM keys.

Exfiltration attacks. We now discuss a limitation of not updating DEM keys during key rotations. For concreteness, consider our KSS scheme in the context of our motivating client and storage service application (described in Section 3). Suppose an attacker compromises for some limited time both the client and the storage service. Then for each ciphertext $(\tilde{C}, C)$ encrypted under a key k_1, the attacker can compute the DEM key $y \oplus r = x$ and exfiltrate it.

Suppose the compromise is cleaned up, and the client immediately generates new keys and rotates all ciphertexts to new secret keys. For the KSS scheme, the resulting ciphertexts will still be later decryptable using the previously exfiltrated DEM keys.

Although a confidentiality issue — the attacker later obtains access to plaintext data they should not have — our UP-IND security notion (and, by implication, the weaker BLMR confidentiality notion) do not capture these attacks. Technically this is because the security game does not allow a challenge ciphertext to be encrypted to a compromised key (or rotated to one). Intuitively, the UP-IND notion gives up on protecting the plaintexts underlying such ciphertexts, as the attacker in the above scenario already had access to the plaintext in the first phase of the attack.

One might therefore argue that this attack is not very important. All of the plaintext data eventually at risk of later decryption was already exposed to the adversary in the first time period because she had access to both the key and ciphertexts. But quantitatively there is a difference: for a given ciphertext an adversary in the first time period can exfiltrate just $|x|$ bits per ciphertext to later recover as much plaintext as she likes, whereas the trivial attack may require exfiltrating the entire plaintext.

The chosen-message attack game of UP-IND does not capture different time periods in which the adversary knows plaintexts in the first time period but "forgets them" in the next. One could explicitly model this, perhaps via a two-stage game with distinct adversaries in each stage, but such games are complex and often difficult to reason about (cf., [RSS11]). We instead follow what we believe is a more intuitive route. This asks that the re-encryption of a ciphertext should leak nothing about the *ciphertext* that was re-encrypted. We use an indistinguishability-style definition to model this. The interpretation of our definition is that any information derivable from a ciphertext (and its secret key) before a re-encryption isn't helpful in attacking the re-encrypted version.

Re-encryption indistinguishability. We formalize this idea via the game shown in Figure 9. The adversary is provided with a left-or-right *re-encryption* oracle, ReLR, instead of the usual left-or-right encryption oracle, in addition to the usual collection of compromised keys, a re-encryption oracle, an encryption oracle, and a rekey token generation oracle.

To avoid trivial wins, the game must disallow the adversary from simply re-encrypting the challenge to a corrupted key. Hence the game uses a table $\mathtt{T}$ to track ciphertexts that have been derived from challenges. Any re-encryption left-or-right query has its response added to the table. Rekey generation applies the rekey token to any relevant existing entries in $\mathtt{T}$ and adds a new entry for the result. A re-encryption query updates the table if there was an entry associated to the queried values.

We associate to parameters κ and t, updatable encryption scheme Π, and UP-REENC$_{\Pi,\kappa,t}$-adversary $\mathcal{A}$ the advantage measure:

$$\mathsf{Adv}^{\text{up-reenc}}_{\Pi,\kappa,t}(\mathcal{A}) = 2 \cdot \Pr\left[\text{UP-REENC}^{\mathcal{A}}_{\Pi,\kappa,t} \Rightarrow \mathsf{true}\right] - 1 \ .$$

Informally, an updatable encryption scheme is UP-REENC secure if no adversary can achieve advantage far from zero given reasonable resources (run time, queries, and number of target keys).

$\underline{\text{UP-REENC}^{\mathcal{A}}_{\Pi,\kappa,t}}$
$b \leftarrow\$ \{0,1\}$
$k_1, \ldots, k_{t+\kappa} \leftarrow\$ \mathsf{KeyGen}()$
$b' \leftarrow\$ \mathcal{A}^{O}(k_{t+1}, \ldots, k_{t+\kappa})$
Return $(b' = b)$

$\underline{\text{ReLR}(i, j, (\tilde{C}_0, C_0), (\tilde{C}_1, C_1))}$
If $\mathsf{Mal}(j)$ or $|C_0| \neq |C_1|$ then
  Return $\perp$
For $\beta \in \{0,1\}$
  $\Delta_{i,j,\tilde{C}_\beta} \leftarrow\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C}_\beta)$
  $(\tilde{C}'_\beta, C'_\beta) \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}_\beta}, C_\beta)$
  If $C'_\beta = \perp$ then Return $\perp$
$\mathtt{T}[j, \tilde{C}'_b] \leftarrow C'_b$
Return $(\tilde{C}'_b, C'_b)$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j)$ and $\mathtt{T}[i, \tilde{C}] \neq \perp$ then Return $\perp$
$\Delta_{i,j,\tilde{C}} \leftarrow\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
If $\mathtt{T}[i, \tilde{C}] \neq \perp$ then
  $(\tilde{C}', C') \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, \mathtt{T}[i, \tilde{C}]))$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow C'$
Return $\Delta_{i,j,\tilde{C}}$

$\underline{\text{Enc}(i, m)}$
Return $\mathsf{Enc}(k_i, m)$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, C))}$
$\Delta_{i,j,\tilde{C}} \leftarrow\$ \mathsf{ReKeyGen}(k_i, k_j, \tilde{C})$
$(\tilde{C}', C') \leftarrow \mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, C))$
If $\mathsf{Mal}(j)$ and $\mathtt{T}[i, \tilde{C}] \neq \perp$ then Return $\tilde{C}'$
If $\mathtt{T}[i, \tilde{C}] \neq \perp$ then $\mathtt{T}[j, \tilde{C}'] \leftarrow C'$
Return $(\tilde{C}', C')$

Fig. 9. The game used to define re-encryption indistinguishability.

$\underline{\text{UP-REENC01}_{\Pi,m}}$
$k_1, k_2 \leftarrow\$ \mathsf{KeyGen}()$
$(\tilde{C}_0, C_0) \leftarrow\$ \mathsf{Enc}(k_1, m)$
$(\tilde{C}_1, C_1) \leftarrow\$ \mathsf{Enc}(k_1, m)$
$\Delta_{1,2,\tilde{C}_1} \leftarrow\$ \mathsf{ReKeyGen}(k_1, k_2, \tilde{C}_1)$
$(\tilde{C}'_1, C'_1) \leftarrow \mathsf{ReEnc}(\Delta_{1,2,\tilde{C}_1}, (\tilde{C}_1, C_1))$
Return $((\tilde{C}_0, C_0), (\tilde{C}'_1, C'_1))$

$\underline{\text{UP-REENC00}_{\Pi,m}}$
$k_1, k_2 \leftarrow\$ \mathsf{KeyGen}()$
$(\tilde{C}_0, C_0) \leftarrow\$ \mathsf{Enc}(k_1, m)$
$\Delta_{1,2,\tilde{C}_0} \leftarrow\$ \mathsf{ReKeyGen}(k_1, k_2, \tilde{C}_0)$
$(\tilde{C}'_0, C'_0) \leftarrow \mathsf{ReEnc}(\Delta_{1,2,\tilde{C}_0}, (\tilde{C}_0, C_0))$
return $((\tilde{C}_0, C_0), (\tilde{C}'_0, C'_0))$

Fig. 10. The re-encryption security experiments from [BLMR15].

Notice that exfiltration attacks as discussed informally above would not be possible for a scheme that meets UP-REENC security. Suppose otherwise, that such exfiltration still worked. Then one could build an UP-REENC adversary that worked as follows. It obtains two encryptions of different messages under a compromised key, calculates the DEM key (or whatever other exfiltration information is useful for later decryption) and then submits the ciphertexts to the ReLR oracle, choosing as target a non-compromised key ($j \leq t$). Upon retrieving the ciphertext, it uses the DEM key (or other exfiltrated data) to decrypt, and checks which message was encrypted. Of course our notion covers many other kinds of attacks, ruling out even re-encryption that allows a single bit of information about the old ciphertext to leak.

BLMR re-encryption security. BLMR introduced a security goal that we will call basic re-encryption indistinguishability.[6] In words, it asks that the distribution of a ciphertext and its re-encryption should be identical to the distribution of a ciphertext and a re-encryption of a distinct ciphertext of the same message.

More formally we have a pair of experiments shown in Figure 10, each parameterized by a message m. Then BLMR require that for all m and all ciphertext pairs $((\tilde{C}, C), (\tilde{C}', C'))$:

$$\left| \Pr[\text{UP-REENC00}_m \Rightarrow ((\tilde{C}, C), (\tilde{C}', C'))] - \Pr[\text{UP-REENC01}_m \Rightarrow ((\tilde{C}, C), (\tilde{C}', C'))] \right| = 0$$

where the probabilities are over the coins used in the experiments.

We feel that this goal misses a number of desirable features captured by our definition. Our definition permits the adversary, for example, to submit *any* pair of ciphertexts to the ReLR oracle. This includes ciphertexts which are encryptions of distinct messages, and even maliciously formed ciphertexts which may not even decrypt correctly. It is simple to exhibit a scheme that meets the BLMR notion but trivially is insecure under ours.[7] On the other hand, suppose a distinguisher exists that can with some probability ϵ distinguish between the outputs of UP-REENC00$_m$ and UP-REENC01$_m$ for some m. Then there exists

[6] BLMR called this ciphertext independence, but we reserve that terminology for schemes that do not require ciphertexts during token generation as per Section 3.

[7] Such a scheme can be constructed by starting with a scheme that satisfies both security notions and adding a "counter" component to ciphertexts that records how many re-encryptions have been performed to obtain that ciphertext; one now

an adversary against our UP-REENC notion which achieves advantage ϵ. This can be seen by the following simple argument. The adversary gets $(\tilde{C}, C) \leftarrow\!\!\$\ \mathsf{Enc}(1, m), (\tilde{C}', C') \leftarrow\!\!\$\ \mathsf{Enc}(1, m)$ and submits the tuple $(1, 2, (\tilde{C}, C), (\tilde{C}', C'))$ to its ReLR oracle and receives a re-encryption of one of the ciphertexts, $(\tilde{C}^*, C^*)$. The adversary then runs the distinguisher on $((\tilde{C}, C), (\tilde{C}^*, C^*))$ and outputs whatever the distinguisher guesses. If the distinguisher is computationally efficient, then so too is the UP-REENC adversary. Thus our UP-REENC notion would be stronger than a computational version of the BLMR notion.

8 Revisiting the BLMR Scheme

The fact that the simple KEM/DEM schemes of Section 5 and Section 6 fail to meet re-encryption security begs the question of finding new schemes that achieve it, as well as UP-IND and UP-INT security. Our starting point is the BLMR construction of an updatable encryption from key-homomorphic PRFs. Their scheme does not (nor did it attempt to) provide integrity guarantees, and so trivially does not meet UP-INT. But before seeing how to adapt it to become suitable as an updatable AE scheme, including whether it meets our stronger notions of UP-IND and UP-REENC security, we first revisit the claims of UP-IND-BI security from [BLMR15].

As mentioned in the introduction, BLMR claim that the scheme can be shown secure, and sketch a proof of UP-IND-BI security. Unfortunately the proof sketch contains a bug, as we explain below. Interestingly, revelation of this bug does not lead to a direct attack on the scheme, and at the same time we could not determine if the proof could be easily repaired. Instead we are able to show that a proof is unlikely to exist.

The main result of this section is a proof showing UP-IND-BI security of the BLMR scheme would imply the existence of a reduction showing that (standard) IND-CPA security implies circular security [BRS03,CL01] for a simple KEM/DEM style symmetric encryption scheme. The latter seems quite unlikely given the known negative results about circular security [ABBC10,CGH12], suggesting that the BLMR scheme is not likely to be provably secure. Towards this result, we first recall some basic tools that BLMR use to build their scheme, then we describe their scheme.

Let $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be an encryption scheme. Let $\text{IND-CPA}_\pi^{\mathcal{A}}$ be the game in which an adversary $\mathcal{A}$ has access to an oracle Enc and must output a bit b'. The game initially chooses a random bit b and a key $k \leftarrow\!\!\$\ \mathcal{KG}$. Enc takes input a pair of bit strings m_0, m_1 and returns $\mathcal{E}_k(m_b)$. The game outputs true when $b' = b$. We require that the adversary's queries always have $|m_0| = |m_1|$. We define the IND-CPA_π advantage of $\mathcal{A}$ as:

$$\mathsf{Adv}_\pi^{\text{ind-cpa}}(\mathcal{A}) = 2 \cdot \Pr\left[\text{IND-CPA}_\pi^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1 \ .$$

The BLMR scheme follows a similar approach to the AE-hybrid scheme, but uses a key-homomorphic PRF in place of regular encryption to enable the data encryption key to also be rotated. Formally, let $\pi_{\text{kem}} = (\mathcal{K}_{\text{kem}}, \mathcal{E}_{\text{kem}}, \mathcal{D}_{\text{kem}})$ be an IND-CPA encryption scheme as above. Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a key-homomorphic PRF where $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are groups, and where the elements of $\mathcal{X}$ can be represented as integers. The BLMR scheme is the tuple of algorithms (KeyGen, Enc, $\mathsf{ReKeyGen}$, ReEnc, Dec) depicted in Figure 11.

Note that encryption in the BMLR scheme consists of a key wrap followed by CTR mode encryption using the wrapped key x and PRF F. For simplicity of presentation we have assumed a padding scheme $\mathsf{pad}_{\mathcal{Y}}, \mathsf{unpad}_{\mathcal{Y}}$ that encodes a message m into a vector of elements of $\mathcal{Y}$.

8.1 Negative Result about Provable UP-IND Security of BLMR

BLMR sketch a proof for the security of this construction in the UP-IND-BI model (as we refer to it). However, the proof misses a subtle point: the interaction with the $\mathsf{ReKeyGen}$ oracle behaves similarly to a decryption oracle and the informal argument given that the IND-CPA security of the KEM is sufficient to argue security is wrong. In fact, the BLMR scheme seems unlikely to be provably secure even in our basic security model. To argue this, we show that proving security of the BLMR scheme implies the 1-circular security of a specific KEM/DEM construction. Figure 12 depicts the security game 1CIRC_π capturing a

exploits the property that any pair of ciphertexts can be input to the ReLR oracle in our UP-REENC game, while only fresh ciphertexts are rotated in the BLMR notion.

$\underline{\mathsf{Enc}(k,m)}$
$x \leftarrow\!\!\$\ \mathcal{K}$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k,x)$
$m_1,\ldots,m_\ell \leftarrow \mathsf{pad}_{\mathcal{Y}}(m)$
For $i = 1$ to ℓ
  $C_i \leftarrow m_i + F(x,i)$
$C \leftarrow C_1\|\cdots\|C_\ell$
Return $(\tilde{C}, C)$

$\underline{\mathsf{ReKeyGen}(k_i,k_j,\tilde{C})}$
$x \leftarrow \mathcal{D}_{\mathrm{kem}}(k_i,\tilde{C})$
$x' \leftarrow\!\!\$\ \mathcal{K}$
$\tilde{C}' \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k_j,x')$
$\Delta_{i,j,\tilde{C}} \leftarrow (\tilde{C}', x' - x)$
Return $\Delta_{i,j,\tilde{C}}$

$\underline{\mathsf{ReEnc}(\Delta_{i,j,\tilde{C}},(\tilde{C},C))}$
$(\tilde{C}',\delta_x) \leftarrow \Delta_{i,j,\tilde{C}}$
$C_1,\ldots,C_\ell \leftarrow C$
For $i = 1$ to ℓ
  $C'_i \leftarrow C_i + F(\delta_x,i)$
$C' \leftarrow C'_1\|\cdots\|C'_\ell$
Return $(\tilde{C}', C')$

$\underline{\mathsf{Dec}(k,(\tilde{C},C))}$
$x \leftarrow \mathcal{D}_{\mathrm{kem}}(k,\tilde{C})$
$C_1,\ldots,C_\ell \leftarrow C$
For $i = 1$ to ℓ
  $m_i \leftarrow C_i - F(x,i)$
$m \leftarrow \mathsf{unpad}_{\mathcal{Y}}(m_1\|\cdots\|m_\ell)$
Return m

Fig. 11. The BLMR scheme built from $\pi_{\mathrm{kem}} = (\mathcal{K}_{\mathrm{kem}}, \mathcal{E}_{\mathrm{kem}}, \mathcal{D}_{\mathrm{kem}})$ and key-homomorphic PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$; KeyGen for the BLMR scheme just runs $\mathcal{K}_{\mathrm{kem}}$.

$\underline{\mathrm{1CIRC}_\pi}$
$b \leftarrow\!\!\$\ \{0,1\}$
$k \leftarrow\!\!\$\ \mathcal{KG}()$
If $b = 0$ then
  $C \leftarrow\!\!\$\ \mathcal{E}(k,k)$
else
  $U \leftarrow\!\!\$\ \{0,1\}^{|k|}$
  $C \leftarrow\!\!\$\ \mathcal{E}(k,U)$
$b' \leftarrow\!\!\$\ \mathcal{A}(C)$
Return $(b = b')$

$\underline{\overline{\mathcal{KG}}()}$
Return $\mathcal{KG}()$

$\underline{\overline{\mathcal{E}}(k,m)}$
Return $\mathcal{E}(k,m)\|\ 0$

$\underline{\overline{\mathcal{D}}(k,C\|\ b)}$
If $b = 0$ then
  Return $\mathcal{D}(k,C)$
else
  Return $k \oplus \mathcal{D}(k,C)$

$\underline{\mathcal{B}^{\mathrm{LR,ReKeyGen}}}$
$U \leftarrow\!\!\$\ \{0,1\}^n$
$(\tilde{C}\|0, C) \leftarrow\!\!\$\ \mathrm{LR}(1,0^n,U)$
$(\tilde{C}'\|0, C') \leftarrow\!\!\$\ \mathrm{ReKeyGen}(1,1,\tilde{C}\|\ 1)$
$b' \leftarrow\!\!\$\ \mathcal{A}(\tilde{C}'\|0, C \oplus C')$
Return b'

Fig. 12. (Left) The 1-circular security game. (Middle) Definition of scheme $\overline{\pi}$ used in the proof of Theorem 5. (Right) UP-IND-BI$_{\overline{\text{S-BLMR}},0,1}$-adversary $\mathcal{B}$ using as a subroutine the adversary $\mathcal{A}$ attacking 1CIRC$_{\overline{\text{S-BLMR}}}$.

simple form of 1-circular security for an encryption scheme $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ which is capable of encrypting plaintexts whose length is the same as the scheme's keys. We associate to scheme π and 1CIRC$_\pi$-adversary $\mathcal{A}$ the advantage measure $\mathsf{Adv}^{\mathrm{1circ}}_{\pi}(\mathcal{A}) = 2 \cdot \Pr[\mathrm{1CIRC}^{\mathcal{A}}_{\pi} \Rightarrow \mathsf{true}] - 1$. We use 1-circular security for simplicity of presentation, but one can generalize our results to longer cycles.

While our main result here (Theorem 5), can be stated for the BLMR scheme as described earlier, for the sake of simplicity we instead give the result for the special case of using a simple one-time pad instead of the key-homomorphic PRF for the DEM component. This is a trivial example of what BLMR call a key-homomorphic PRG, and their theorem statement covers this construction as well. We will show that proving security for this special case is already problematic, and this therefore suffices to call into question their (more general) theorem. This simplified BLMR scheme, denoted S-BLMR in what follows, has as its encryption algorithm $\mathsf{Enc}(k,m) = (\mathcal{E}(k,r), r \oplus m)$ where, as usual, $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is an IND-CPA secure symmetric key encryption scheme. We assume $|m| = n$. We then also have, for S-BLMR, $\mathsf{ReKeyGen}(k_1,k_2,\tilde{C}) = (\mathcal{E}(k_2,r'), r' \oplus \mathcal{D}(k_1,\tilde{C}))$ and $\mathsf{ReEnc}((\tilde{C}',\delta_r),(\tilde{C},C)) = (\tilde{C}', C \oplus \delta_r)$. We have the following theorem:

Theorem 5. *Let $\pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. We give a symmetric encryption scheme $\overline{\pi} = (\overline{\mathcal{KG}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ in Figure 12 such that:*

(1) (IND-CPA$_\pi$ *implies* IND-CPA$_{\overline{\pi}}$) *For any* IND-CPA$_{\overline{\pi}}$*-adversary $\mathcal{A}_1$, we give an* IND-CPA$_\pi$*-adversary $\mathcal{B}_1$ such that $\mathsf{Adv}^{\text{ind-cpa}}_{\overline{\pi}}(\mathcal{A}_1) \le \mathsf{Adv}^{\text{ind-cpa}}_{\pi}(\mathcal{B}_1)$. Adversary $\mathcal{B}_1$ runs in time that of $\mathcal{A}_1$ and makes the same number of queries.*

(2) (UP-IND-BI$_{\overline{\text{S-BLMR}},0,1}$ *implies* 1CIRC$_{\overline{\text{S-BLMR}}}$) *Let* $\overline{\text{S-BLMR}}$ *denote the simplified BLMR updatable encryption instantiated using $\overline{\pi}$. Then for any* 1CIRC$_{\overline{\text{S-BLMR}}}$*-adversary $\mathcal{A}_2$, we give an* UP-IND-BI$_{\overline{\text{S-BLMR}},0,1}$*-adversary $\mathcal{B}_2$ such that $\mathsf{Adv}^{\mathrm{1circ}}_{\overline{\text{S-BLMR}}}(\mathcal{A}_2) \le \mathsf{Adv}^{\text{up-ind-bi}}_{\overline{\text{S-BLMR}},0,1}(\mathcal{B}_2)$. Adversary $\mathcal{B}_2$ runs in time that of $\mathcal{A}_2$ plus $\mathcal{O}(1)$ overhead and makes a single query.*

(3) (1CIRC$_{\overline{\text{S-BLMR}}}$ *implies* 1CIRC$_{\text{S-BLMR}}$) *For any* 1CIRC$_{\text{S-BLMR}}$ *adversary $\mathcal{A}_3$, we give a* 1CIRC$_{\overline{\text{S-BLMR}}}$*-adversary $\mathcal{B}_3$ such that $\mathsf{Adv}^{\mathrm{1circ}}_{\text{S-BLMR}}(\mathcal{A}_3) \le \mathsf{Adv}^{\mathrm{1circ}}_{\overline{\text{S-BLMR}}}(\mathcal{B}_3)$. Adversary $\mathcal{B}_3$ runs in time that of $\mathcal{A}$ plus $\mathcal{O}(1)$ overhead.*

Before giving the proof, we first point out the implications of this (somewhat complicated) theorem statement. Basically we have the following sequence of implications:

$$\text{IND-CPA}_{\pi} \Rightarrow \text{IND-CPA}_{\overline{\pi}} \Rightarrow \text{UP-IND-BI}_{\overline{\text{S-BLMR}},0,1} \Rightarrow \text{1CIRC}_{\overline{\text{S-BLMR}}} \Rightarrow \text{1CIRC}_{\text{S-BLMR}}$$

where each arrow implies the ability to give an efficient, concrete reduction showing that any adversary against the right-hand-side game implies an equally efficient adversary against the left-hand-side-game. The first, third, and fourth implications are due to Theorem 5 parts (1), (2), and (3). The second implication is the BLMR claim.

The result is relative, showing that a proof of BLMR's claim implies a reduction (via the chain above) between circular security for the particular scheme S-BLMR and IND-CPA security for π. It is possible that this reduction exists, however it seems unlikely. Existing counter-examples show IND-CPA schemes that are not circular-secure [KRW15]. While these counter-examples do not have the same form as the specific scheme under consideration, it may be that one can build a suitable counter-example with additional effort. In any case, we take this as strong evidence that correcting the proof from BLMR will be difficult, if not impossible.

Proof. We start by introducing $\overline{\pi}$, a slight variant of π. It is detailed in Figure 12. It adds a bit to the ciphertext[8] that is read during decryption: if the bit is 1 then decryption outputs the secret key xor'd with the plaintext.

The first and third parts of the theorem are straightforward reductions and we omit them for brevity. The main point is that the artificially introduced weakness of $\overline{\pi}$ only impacts settings that allow chosen-ciphertext attacks.

We now turn to the second part of the theorem. Let $\overline{\text{S-BLMR}} = (\overline{\mathsf{KeyGen}}, \overline{\mathsf{Enc}}, \overline{\mathsf{ReKeyGen}}, \overline{\mathsf{ReEnc}}, \overline{\mathsf{Dec}})$ be the simplified BLMR scheme instantiated using $\overline{\pi}$, so that $\overline{\mathsf{Enc}}(k, m) = (\overline{\mathcal{E}}(k, r), r \oplus m)$ and $\overline{\mathsf{ReKeyGen}}(k_1, k_2, C) = (\overline{\mathcal{E}}(k_2, r'), r' \oplus \overline{\mathcal{D}}(k_1, C))$. Since $\overline{\pi}$ is IND-CPA, the security claim of BLMR implies that $\overline{\text{S-BLMR}}$ is UP-IND-BI. We will now show that UP-IND-BI security of $\overline{\text{S-BLMR}}$ implies the 1-circular security of $\overline{\text{S-BLMR}}$. To this end, let $\mathcal{A}$ be a 1-circular adversary against $\overline{\text{S-BLMR}}$. Then we build an adversary $\mathcal{B}$ against the UP-IND-BI security of $\overline{\text{S-BLMR}}$. It is shown in Figure 12. The adversary makes an LR query on the message 0^n and a unformly random message U. If the UP-IND-BI challenge bit is 0 then it gets back a ciphertext $C_0 = (\mathcal{E}(k_1, r) \| 0, r)$ and if it is 1 then it gets back $C_1 = (\mathcal{E}(k_1, r) \| 0, r \oplus U)$. Next it queries its ReKeyGen oracle on the first component of the returned ciphertext but with the trailing bit switched to 1, asking for a rekey token for rotating from k_1 back to k_1. The value returned by this query is equal to the pair $(\mathcal{E}(k_1, r') \| 0, r' \oplus k_1 \oplus r)$. By XOR'ing the second component here with the second component returned from the LR query the adversary constructs a ciphertext that is the $\overline{\text{S-BLMR}}$ encryption of k_1 under itself when the UP-IND-BI challenge bit is 0 and the $\overline{\text{S-BLMR}}$ encryption of a uniformly random message under k_1 when the UP-IND-BI challenge bit is 1. Adversary $\mathcal{B}$ runs the 1-circular adversary $\mathcal{A}$ for the scheme $\overline{\text{S-BLMR}}$ on the final ciphertext and outputs whatever $\mathcal{A}$ outputs. □

9 An Updatable AE Scheme with Re-encryption Indistinguishability

We first point out that one can avoid the issues raised in Section 8 by replacing the IND-CPA KEM with a proper AE scheme. This does not yet, however, address integrity of the full encryption scheme. We therefore provide a new construction — which which we refer to as ReCrypt. It is detailed in Figure 13. It uses an AE scheme $\pi_{\text{kem}} = (\mathcal{K}_{\text{kem}}, \mathcal{E}_{\text{kem}}, \mathcal{D}_{\text{kem}})$, a key-homomorphic PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, and a hash function $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$. Here $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ are groups. It also uses a padding scheme $(\mathsf{pad}_{\mathcal{Y}}, \mathsf{unpad}_{\mathcal{Y}})$ which is a pair of algorithms. Algorithm $\mathsf{pad}_{\mathcal{Y}}$ takes input a bit string (the message) and outputs a sequence of ℓ elements of $\mathcal{Y}$. Algorithm $\mathsf{unpad}_{\mathcal{Y}}$ takes input a sequence of elements of $\mathcal{Y}$ and outputs a message, or an error symbol $\perp$. We require that the scheme is correct, meaning that $\mathsf{unpad}_{\mathcal{Y}}(\mathsf{pad}_{\mathcal{Y}}(m)) = m$ for all bit strings m, and that $\mathsf{pad}_{\mathcal{Y}}$ must be tidy, meaning that for all bit strings m, m' such that $|m| = |m'|$, we have $|\mathsf{pad}_{\mathcal{Y}}(m)| = |\mathsf{pad}_{\mathcal{Y}}(m')|$. Furthermore, to meet our integrity requirement, we need $\mathsf{unpad}_{\mathcal{Y}}(y) \neq \mathsf{unpad}_{\mathcal{Y}}(y')$ for all distinct sequences of elements $y, y' \in \mathcal{Y}^+$. We discuss specific instantiations of the padding scheme in Section 10. We also assume that elements of $\mathcal{X}$ can be represented by integers (as in the BLMR scheme), and discuss this assumption further in the next section.

[8] Notice that this scheme is not tidy in the sense of [NRS14]. While that doesn't affect the implications of our analysis – BLMR make no assumptions about tidiness – finding a tidy counter-example is an interesting open question.

$\underline{\mathsf{Enc}(k, m)}$
$x, r \leftarrow\!\!\$\ \mathcal{K}$
$m_1, \ldots, m_\ell \leftarrow \mathsf{pad}_{\mathcal{Y}}(m)$
For $i = 1$ to ℓ
  $C_i \leftarrow m_i + F(x, i)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k, (r + x) \parallel \tau)$
$C \leftarrow C_1 \| \cdots \| C_\ell$
Return $(\tilde{C}, (r, C))$

$\underline{\mathsf{ReKeyGen}(k_i, k_j, \tilde{C})}$
$(y \parallel \tau) \leftarrow \mathcal{D}_{\mathrm{kem}}(k_i, \tilde{C})$
If $(y \parallel \tau) = \perp$ then
  Return $\perp$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{dem}}$
$y' \leftarrow y + x' + r'$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\!\$\ \mathcal{E}_{\mathrm{kem}}(k_j, (y' \parallel \tau'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\mathsf{ReEnc}(\Delta_{i,j,\tilde{C}}, (\tilde{C}, (r, C)))}$
$(\tilde{C}', x', r') \leftarrow \Delta_{i,j,\tilde{C}}$
$C_1, \ldots, C_\ell \leftarrow C$
For $i = 1$ to ℓ
  $C'_i \leftarrow C_1 + F(x', i)$
$C' \leftarrow C_1 \| \cdots \| C_\ell$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\mathsf{Dec}(k, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \mathcal{D}_{\mathrm{kem}}(k, \tilde{C})$
If $(y \parallel \tau) = \perp$ then
  Return $\perp$
$C_1, \ldots, C_\ell \leftarrow C$
$x \leftarrow y - r$
For $i = 1$ to ℓ
  $m_i \leftarrow C_i - F(x, i)$
$m \leftarrow \mathsf{unpad}_{\mathcal{Y}}(m_1 \| \cdots \| m_\ell)$
If $\tau - F(x, 0) \neq \mathcal{H}(m)$ then
  Return $\perp$
Return m

Fig. 13. Algorithms for the ReCrypt updatable encryption scheme; KeyGen for ReCrypt simply runs $\mathcal{K}_{\mathrm{kem}}$.

The construction works as follows. As with KSS, the AE scheme is used to encrypt a share of the DEM key as well as an authentication tag. The latter we must construct in a specific way to allow DEM keys to be rotated: we mask a hash of the message with the key-homomorphic PRF. This is somewhat like a Carter-Wegman MAC [WC81], though we use a fixed hash function as opposed to a (keyed) universal hash function and won't rely on traditional MAC security. Instead we will require a collision-resistance-style property to show ciphertext integrity of the full construction. We encrypt the message using a PRF-generated pad in counter mode, as in the BLMR scheme. The final ciphertext consists of the combination of the AE ciphertext header, the counter-mode output, and the other share of the DEM key.

To perform rotations, we refresh the shares by picking a new random value r' and updating the first share to be $r + r'$ (with addition over $\mathcal{K}$). Refreshing the share is important to prevent attacks that use revealed shares from before and after a rotation to reveal the DEM key. We similarly refresh the DEM key, setting the post-rotation DEM key to be equal to $x + x'$ for some fresh x'. The latter is a bit different compared to the BLMR scheme, where the new DEM key was simply set to be x'. The reason for this difference is that the secret sharing prevents, by design, rekey token generation from knowing x, and thus the algorithm cannot compute $x' - x$. Our solution therefore just adds a fresh random value into the key, having key rotation serve as an accumulator. We'll prove that this suffices to refresh the key. Intuitively it follows because $x + x'$ is uniform even if x is known, assuming x' is freshly and uniformly chosen. The combination of the rekey token generation plus re-encryption essentially encrypts the previous ciphertext and tag with a freshly keyed one-time real-or-random encryption scheme.

To make some of our description of encryption and decryption more compact, we introduce some notation related to repeated applications of F. Fixing a group $\mathcal{Y}$, let $\ell(m)$ be the function that maps m to the number of group elements needed to encode $\mathsf{pad}_{\mathcal{Y}}(m)$. Let $\hat{F}^i(x)$ be equal to $F(x, 1), F(x, 2), \ldots, F(x, i)$. Thus, $\hat{F}^{\ell(m)}(x)$ outputs the vector of group elements used as pad in encryption of the message m. Letting C be a vector of group elements, we denote by $|C|$ the dimension of that vector. Thus $\hat{F}^{|C|}(x)$ generates a vector of group elements of length the number of elements in C. Finally we assume vector addition of group elements is done component wise in the usual way, i.e., $C + \hat{F}^{|C|}(x)$ adds each component of C to a component of the generated pad.

Assuming that the underlying AE scheme π_{kem} is correct, so too is ReCrypt. To see this, consider any message m and any sequence of (legitimate) keys $k_1, \ldots, k_T$. Let $(\tilde{C}_1, (r_1, C_1))$ be the output of running $\mathsf{Enc}(k_1, m)$. Then we have that the ciphertext resulting from $T - 1$ applications of rekey token generation and re-encryption has ciphertext header:

$$\mathcal{E}_{\mathrm{kem}}\left(k_T, \left(\sum_{i=1}^{T}(x_i + r_i)\right) \Big|\Big| \left(\mathcal{H}(m) + \sum_{i=1}^{T} F(x_i, 0)\right)\right),$$

and ciphertext body:

$$\left(\sum_{i=1}^{T} r_i\right) \Big|\Big| \left(\mathsf{pad}_{\mathcal{Y}}(m) + \sum_{i=1}^{T}\left(\hat{F}^{\ell(m)}(x_i)\right)\right) .$$

This ciphertext correctly decrypts under k_T, as it will recover the shares and authentication tag from the header, then subtract the sum of all randomness values r_i to get an accumulated DEM key $\sum_{i=1}^{T} x_i$. The key homomorphic property of F ensures that the recomputation of the tag and the pad matches those generated over the course of the re-encryptions.

In the remainder of this section we show that the new scheme meets our strongest security notions for updatable encryption.

9.1 Message Confidentiality of ReCrypt

To analyze security we start with message confidentiality. We make our proof more modular (and more similar to the proof of UP-IND for KSS) by defining an AE scheme associated to the DEM underlying ReCrypt. Formally, let $\pi_{\mathrm{dem}} = (\mathcal{K}_{\mathrm{dem}}, \mathcal{E}_{\mathrm{dem}}, \mathcal{D}_{\mathrm{dem}})$ be the AE scheme associated with $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$ that works as follows. Key generation for π_{dem} simply samples a uniform value from $\mathcal{K}$. The encryption algorithm $\mathcal{E}_{\mathrm{dem}}$ is deterministic: on input a key $x \in \mathcal{K}$ and message $m \in \{0,1\}^*$ it computes $C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{|\mathsf{pad}_{\mathcal{Y}}(m)|}(x)$ and $\tau \leftarrow \mathcal{H}(m) + F(x,0)$, and outputs (C,τ). On input a key $x \in \mathcal{K}$ and ciphertext (C,τ), the decryption algorithm $\mathcal{D}_{\mathrm{dem}}$ first computes $m \leftarrow C - \hat{F}^{|C|}(x)$, then computes $\tau' \leftarrow \mathcal{H}(m) + F(x,0)$, and outputs m if $\tau' = \tau$ and otherwise outputs $\perp$. The following lemma establishes that this scheme achieves one-time real-or-random security.

Lemma 1. *Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$. Let π_{dem} be the AE scheme associated with $F, \mathcal{H}$. Then for any* OT-ROR$_{\pi_{dem}}$ *adversary $\mathcal{A}$ making at most q queries on messages whose maximum length after applying $\mathsf{pad}_{\mathcal{Y}}$ is at most σ, we give an adversary $\mathcal{B}$ below such that: $\mathsf{Adv}^{\text{ot-ror}}_{\pi_{dem}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{mu-prf}}_{F}(\mathcal{B})$, where $\mathcal{B}$ runs in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q\sigma)$ overhead and where it makes at most $q\sigma$ queries.*

The proof follows from a straightforward reduction to the multi-use PRF security of F that we omit for brevity. The following theorem captures the UP-IND security achieved by ReCrypt. While we use the lemma above, the reduction to the security of π_{dem} is not black-box due to the extra rotation functionality needed from it.

Theorem 6 (UP-IND **Security of** ReCrypt**).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ be an AE scheme, $F : \mathcal{X} \to \mathcal{Y}$ be a key-homomorphic PRF, $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$ be a hash function, and π_{dem} the AE scheme associated with $F, \mathcal{H}$. Let* ReCrypt *be the updatable AE scheme as defined in Figure 13. Let $t > 0$ and $\kappa \geq 0$. Then for any* UP-IND$_{\mathrm{ReCrypt},\kappa,t}$ *adversary $\mathcal{A}$ making at most q queries of length at most σ, we give adversaries $\mathcal{B}, \mathcal{C}$ such that*

$$\mathsf{Adv}^{\text{up-ind}}_{\mathrm{ReCrypt},\kappa,t}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem}}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{dem}}(\mathcal{C}) \ .$$

Adversaries $\mathcal{B}, \mathcal{C}$ each run in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q + t + \kappa)$ overhead. Adversary $\mathcal{B}$ makes at most q queries. Adversary $\mathcal{C}$ makes at most q queries of length at most σ.

By combining Theorem 6 and Lemma 1 we arrive at a security reduction of ReCrypt to that of π_{kem} and the PRF F. The proof works essentially identically to that of KSS, following the same main three steps. First we replace KEM encryption with its ideal counterpart. We then argue that the secret sharing used ensures that the ciphertext header returned by an invalid re-encryption query reveals nothing about challenge DEM keys. Next we argue that no sequence of queries will reveal the KEM-encrypted key shares generated during challenge queries, and then we reduce to the one-time, real-or-random security of the DEM to move to a game in which the adversary's view is independent of the challenge bit.

The only distinction is accounting in the games for the extra functionality used in ReKeyGen and ReEnc for this scheme, i.e., picking fresh DEM keys and rotating tags and ciphertext portions. These do not impact the security argument (they will be used in the next proof, of re-encryption indistinguishability). However, for completeness we give the full proof of Theorem 6 in Appendix A.

9.2 Re-encryption Indistinguishability of ReCrypt

We turn to proving that ReCrypt meets our re-encryption indistinguishability notion, as captured by the following theorem. The structure of the argument is very similar to how we have argued UP-IND security,

except that we ultimately reduce to the one-time, real-or-random security of the DEM re-encryption process. In other words, the process of rotating the DEM tag and ciphertext portion amounts to freshly encrypting the tag and ciphertext portion.

To make this explicit, we extract from ReCrypt the IND-CPA secure encryption scheme implicitly underlying re-encryption. Formally, let $\pi_{\mathrm{re}} = (\mathcal{K}_{\mathrm{re}}, \mathcal{E}_{\mathrm{re}}, \mathcal{D}_{\mathrm{re}})$ be the symmetric encryption scheme that works as follows. Key generation algorithm $\mathcal{K}_{\mathrm{re}}$ outputs a random draw from $\mathcal{K}$. Encryption algorithm $\mathcal{E}_{\mathrm{re}}$ takes input a key x' and a pair $(C, \tau) \in \mathcal{Y}^+ \times \mathcal{Y}$, i.e., a vector C of one or more elements of $\mathcal{Y}$ and a single group element τ. It outputs $((C + \hat{F}^{|C|}(x')), (\tau' + F(x', 0)))$. We won't need the decryption algorithm $\mathcal{D}_{\mathrm{re}}$ for our analysis, but for completeness it takes input key x' and a pair (C', τ') and outputs $((C' - \hat{F}^{|C|}(x')), (\tau' - F(x', 0)))$.

The following lemma shows that this encryption scheme is one-time, real-or-random secure assuming F is a good PRF.

Lemma 2. *Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ and π_{re} be the encryption scheme based on F as defined above. Then for any* OT-ROR$_{\pi_{re}}$ *adversary $\mathcal{A}$ making at most q queries with maximum length σ group elements, we give an adversary $\mathcal{B}$ below such that $\mathsf{Adv}^{\text{ot-ror}}_{\pi_{re}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{mu-prf}}_{F}(\mathcal{B})$, where $\mathcal{B}$ runs in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q\sigma)$ overhead and where it makes at most $q\sigma$ queries.*

The proof is straightforward and we omit it for brevity. We can now state the main theorem, which captures the UP-REENC security of ReCrypt.

Theorem 7 (UP-REENC **Security of** ReCrypt**).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ be an AE scheme, $F : \mathcal{X} \to \mathcal{Y}$ be a key-homomorphic PRF, and $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$ be a hash function, and π_{re} the encryption scheme built from F as described above. Let* ReCrypt *be the updatable AE scheme as defined in Figure 13. Let $t > 0$ and $\kappa \geq 0$. Then for any* UP-REENC$_{\mathrm{ReCrypt},\kappa,t}$ *adversary $\mathcal{A}$ making at most q queries of length at most σ, we give adversaries $\mathcal{B}, \mathcal{C}$ in the proof below such that*

$$\mathsf{Adv}^{\text{up-ind}}_{\mathrm{ReCrypt},\kappa,t}(\mathcal{A}) \leq 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem}}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{re}}(\mathcal{C}) \ .$$

Adversary $\mathcal{B}$ makes at most q queries. Adversary $\mathcal{C}$ makes at most q queries with messages of length at most $|\mathsf{pad}_{\mathcal{Y}}(m)| + 1$ elements of $\mathcal{Y}$ for a σ-bit string m. Each run in time at most that of $\mathcal{A}$ plus a $\mathcal{O}(q + t + \kappa)$ overhead.

By combining the theorem with Lemma 2 we obtain a reduction from ReCrypt's re-encryption indistinguishability to the KEM's security and the PRF's security. The proof of Theorem 7 is very similar to the proof of message indistinguishability for ReCrypt and, in turn, of message indistinguishability for KSS. Nevertheless we give it below for completeness.

Proof. Our proof uses a sequence of games. For compactness, we let Π denote the scheme ReCrypt. Game G_0 (Figure 14, boxed statements omitted) is the UP-REENC$_{\Pi,\kappa,t}$ game. In this game, and in all subsequent ones, we omit the main procedure to save space. That procedure picks a random bit b, generates secret keys $k_1, \ldots, k_{t+\kappa} \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{kem}}$, runs $\mathcal{A}$ with input $k_{t+1}, \ldots, k_{t+\kappa}$ and with access to the oracles, and returns true if $\mathcal{A}$'s output b' is equal to b.

Game G_1 (Figure 14, boxed statements included) replaces the KEM encryption when used with honest keys with randomly chosen ciphertexts and table look-ups to implement decryption. To bound the transition between G_0 and G_1 we use a reduction to the AE security of the KEM. In detail, let $\mathcal{B}$ be the MU-ROR-AE$_{\pi,t}$ adversary that works exactly like game G_0 (and G_1) except that: (1) uses of $\mathcal{E}_{\mathrm{kem}}(k_i, m)$ with $i \leq t$ are replaced with a query to $\mathcal{B}$'s encryption oracle on (i, m) and (2) uses of $\mathcal{D}_{\mathrm{kem}}(k_i, \tilde{C})$ with $i \leq t$ are replaced with a query to $\mathcal{B}$'s decryption oracle on $(i, \tilde{C})$. By construction, it holds that MU-ROR-AE0$^{\mathcal{B}}_{\pi,t} = G_0^{\mathcal{A}}$ and MU-ROR-AE1$^{\mathcal{B}}_{\pi,t} = G_1^{\mathcal{A}}$. Thus,

$$\Pr[\, G_0 \Rightarrow \mathsf{true} \,] \leq \Pr[\, G_1 \Rightarrow \mathsf{true} \,] + \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi,t}(\mathcal{B}) \ .$$

We now will make transitions so that the transcript observed by the adversary is independent of the challenge bit b. This will involve: (1) showing that headers returned in response to invalid ReEnc queries are indistinguishable from encryptions of a random DEM key and the tag (because of the secret-sharing of DEM keys and refreshing of them during re-encryption); (2) the y values generated during challenge queries are never revealed to the adversary (which we argue by induction using the rules preventing certain rekey or re-encryption queries); and (3) applying the OT-ROR security of π_{re} to argue that the re-encrypted ciphertext leaks nothing about the challenge bit b.

$\underline{\text{Enc}(i,m)}$ G_0 , $\boxed{G_1}$
$x, r \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{\ell(m)}(x)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\$\ \text{E}_{\text{kem}}(i, (x + r) \parallel \tau)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\bot$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $(r, C) \leftarrow \mathtt{T}[i, \tilde{C}]$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C + \hat{F}^{|C|}(x'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C' \leftarrow C + \hat{F}^{|C|}(x')\ ;\ \tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\tilde{C}'$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C')$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\text{ReLR}(i, j, (\tilde{C}_0, (r_0, C_0)), (\tilde{C}_1, (r_1, C_1)))}$
If $\mathsf{Mal}(j)$ or $|C_0| \neq |C_1|$ then Return $\bot$
$(y_0 \parallel \tau_0) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C}_0)$
$(y_1 \parallel \tau_1) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C}_1)$
If $(y_0 \parallel \tau_0) = \bot$ or $(y_1 \parallel \tau_1) = \bot$ then
  Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C' \leftarrow C_b + \hat{F}^{|C_0|}(x')\ ;\ \tau' \leftarrow \tau_b + F(x', 0)$
$y' \leftarrow y_b + x' + r'$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, y' \parallel \tau')$
$\mathtt{T}[j, \tilde{C}'] \leftarrow (r_b + r', C')$
Return $(\tilde{C}', (r_b + r', C'))$

$\underline{\text{E}_{\text{kem}}(i, m)}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{E}_{\text{kem}}(k_i, m)$
$\tilde{C} \leftarrow\!\$\ \mathcal{E}_{\text{kem}}(k_i, m)$
$\boxed{\tilde{C} \leftarrow\!\$\ \{0,1\}^{|\tilde{C}|}\ ;\ \mathtt{C}[i, \tilde{C}] \leftarrow m}$
Return $\tilde{C}$

$\underline{\text{D}_{\text{kem}}(i, \tilde{C})}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$m \leftarrow \mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$\boxed{m \leftarrow \mathtt{C}[i, \tilde{C}]}$
Return m

$\underline{\text{Enc}(i,m)}$ G_3 , $\boxed{G_4}$
$x, r \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{\ell(m)}(x)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\$\ \text{E}_{\text{kem}}(i, (x + r) \parallel \tau)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\bot$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $(r, C) \leftarrow \mathtt{T}[i, \tilde{C}]$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C + \hat{F}^{|C|}(x'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\tau' \leftarrow \tau + F(x', 0)$
  $\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, r' \parallel \tau')$
  Return $\tilde{C}'$
$C' \leftarrow C + \hat{F}^{|C|}(x')\ ;\ \tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C')$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\text{ReLR}(i, j, (\tilde{C}_0, (r_0, C_0)), (\tilde{C}_1, (r_1, C_1)))}$
If $\mathsf{Mal}(j)$ or $|C_0| \neq |C_1|$ then Return $\bot$
$(y_0 \parallel \tau_0) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C}_0)$
$(y_1 \parallel \tau_1) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C}_1)$
If $(y_0 \parallel \tau_0) = \bot$ or $(y_1 \parallel \tau_1) = \bot$ then
  Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C' \leftarrow C_b + \hat{F}^{|C_0|}(x')\ ;\ \tau' \leftarrow \tau_b + F(x', 0)$
$y' \leftarrow y + x' + r'$; $\boxed{y' \leftarrow 0^{|r|}}$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, y' \parallel \tau')$
$\mathtt{T}[j, \tilde{C}'] \leftarrow (r_b + r', C')$
Return $(\tilde{C}', (r_b + r', C'))$

$\underline{\text{E}_{\text{kem}}(i, m)}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{E}_{\text{kem}}(k_i, m)$
$\tilde{C} \leftarrow\!\$\ \{0,1\}^{|\tilde{C}|}\ ;\ \mathtt{C}[i, \tilde{C}] \leftarrow m$
Return $\tilde{C}$

$\underline{\text{D}_{\text{kem}}(i, \tilde{C})}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$m \leftarrow \mathtt{C}[i, \tilde{C}]$
Return m

$\underline{\text{Enc}(i,m)}$ G_5 , $\boxed{G_6}$
$x, r \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{\ell(m)}(x)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\$\ \text{E}_{\text{kem}}(i, (x + r) \parallel \tau)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\bot$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $(r, C) \leftarrow \mathtt{T}[i, \tilde{C}]$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C + \hat{F}^{|C|}(x'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\tau' \leftarrow\!\$\ \mathcal{Y}$; $\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, r' \parallel \tau')$
  Return $\tilde{C}'$
$C' \leftarrow C + \hat{F}^{|C|}(x')\ ;\ \tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C')$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\text{ReLR}(i, j, (\tilde{C}_0, (r_0, C_0)), (\tilde{C}_1, (r_1, C_1)))}$
If $\mathsf{Mal}(j)$ or $|C_0| \neq |C_1|$ then Return $\bot$
$(y_0 \parallel \tau_0) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C}_0)$
$(y_1 \parallel \tau_1) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C}_1)$
If $(y_0 \parallel \tau_0) = \bot$ or $(y_1 \parallel \tau_1) = \bot$ then
  Return $\bot$
$x', r' \leftarrow\!\$\ \mathcal{K}_{\text{dem}}$
$C' \leftarrow C_b + \hat{F}^{|C_0|}(x')\ ;\ \tau' \leftarrow \tau_b + F(x', 0)$
$\boxed{C' \leftarrow\!\$\ \mathcal{Y}^{|C_0|}\ ;\ \tau' \leftarrow\!\$\ \mathcal{Y}}$
$\tilde{C}' \leftarrow\!\$\ \text{E}_{\text{kem}}(j, 0^{|r'|} \| \tau')$
$\mathtt{T}[j, \tilde{C}'] \leftarrow (\ r'\ , C')$
Return $(\tilde{C}', (\ r'\ , C'))$

$\underline{\text{E}_{\text{kem}}(i, m)}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{E}_{\text{kem}}(k_i, m)$
$\tilde{C} \leftarrow\!\$\ \{0,1\}^{|\tilde{C}|}\ ;\ \mathtt{C}[i, \tilde{C}] \leftarrow m$
Return $\tilde{C}$

$\underline{\text{D}_{\text{kem}}(i, \tilde{C})}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$m \leftarrow \mathtt{C}[i, \tilde{C}]$
Return m

Fig. 14. Games for proof of UP-REENC for ReCrypt.

Towards this we start by modifying how ReEnc returns headers in the case of an invalid query (one for which $\mathtt{T}[i,\tilde{C}] \neq \perp$ and $j > t$). Notice that in game G_1, in the case that just the header is returned in response to a ReEnc query, the header's plaintext portion $y + x' + r'$ is independent of $y + x'$ because r' is uniform and not used elsewhere in the game. We make this explicit in game G_2 (Figure 14, boxed statement omitted) which has ReEnc return a $\mathcal{E}_{\mathrm{kem}}(k_j, r' \| \tau')$ instead of $\mathcal{E}_{\mathrm{kem}}(k_j, (y + x' + r') \| \tau')$. Game G_2 also dispenses with the now redundant lines of code from G_1's handling of $\mathrm{E}_{\mathrm{kem}}$, $\mathrm{D}_{\mathrm{kem}}$. We have that $\Pr[G_1^{\mathcal{A}} \Rightarrow \mathsf{true}] = \Pr[G_2^{\mathcal{A}} \Rightarrow \mathsf{true}]$. Note that this step of the proof relies on the secret sharing of the DEM key — without it we would not be able to argue independence and, in fact, the returned ciphertext header would reveal the DEM key.

In game G_3, we replace the DEM key share y in ReLR with a constant value. We will argue that

$$\Pr\left[\, G_2^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] = \Pr\left[\, G_3^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right], \tag{3}$$

by arguing that no values returned to the adversary depend on y. Intuitively, the reason is that while y may affect other entries of $\mathtt{C}$, via rekey or re-encryption queries, the rules about invalid queries combined with our earlier game transition to G_2 ensure that these plaintext values never impact any values returned to the adversary.

To argue this, consider any query to ReLR on index i^* that produces an entry $\mathtt{C}[i^*, \tilde{C}^*] = y \| \tau$. Necessarily $\mathtt{T}[i^*, \tilde{C}^*] \neq \perp$. We perform an inductive argument over the subsequent queries to show the following condition holds after each query: the only values in the game dependent on y are entries in $\mathtt{C}[j, \tilde{C}]$ for some $j, \tilde{C}$ for which $\mathtt{T}[j, \tilde{C}] \neq \perp$. In particular, no values returned to the adversary are dependent. By dependent we mean that the values are a function of y and (possibly) other variables defined in the game. Below we refer to entries $j, \tilde{C}$ for which $\mathtt{C}[j, \tilde{C}]$ depends on y as dependent entries.

As base case consider that the ReLR query was the last query. Then the condition holds by inspection of the code of ReLR in game G_2. Now consider an arbitrary later query, and assume that the condition holds at the end of the previous query. We have the following cases broken down by the type of query.

(1) The query is to Enc. Then no dependent entries are accessed, and so the condition trivially holds after the query.

(2) The query is to ReKeyGen on a triple $i, j, \tilde{C}$. If $i, \tilde{C}$ is not a dependent entry, then the condition trivially holds after the query. Otherwise, by the inductive hypothesis $\mathtt{T}[i, \tilde{C}] \neq \perp$ and so for $\mathtt{C}[i, \tilde{C}]$ to be accessed, necessarily $\mathsf{Hon}(j)$. Then the procedure generates a new dependent plaintext submitted to $\mathrm{E}_{\mathrm{kem}}$ for $\mathsf{Hon}(j)$, thereby adding a new dependent entry $\mathtt{C}[j, \tilde{C}']$. Because $\mathtt{T}[i, \tilde{C}] \neq \perp$ it holds that $\mathtt{T}[j, \tilde{C}']$ is set to a value other than $\perp$. None of the returned values depend on either dependent entry. Thus the condition holds after the query completes.

(3) The query is to ReEnc on a triple $i, j, (\tilde{C}, (r, C))$. If $i, \tilde{C}$ is not a dependent entry, then the condition trivially holds after the query. Otherwise, by the inductive hypothesis $\mathtt{T}[i, \tilde{C}] \neq \perp$. If $\mathsf{Mal}(j)$ then no new dependent entries are generated (the KEM plaintext in this case using r' instead of $y \oplus r'$). If $\mathsf{Hon}(j)$, then the procedure generates a new dependent plaintext submitted to $\mathrm{E}_{\mathrm{kem}}$ for $\mathsf{Hon}(j)$, adding a new dependent entry $\mathtt{C}[j, \tilde{C}']$. Because $\mathtt{T}[i, \tilde{C}] \neq \perp$ it holds that $\mathtt{T}[j, \tilde{C}']$ is set to a value other than $\perp$. None of the returned values depend on either dependent entry. Thus the condition holds after the query completes.

Thus by induction the condition holds for all queries, and we have justified (3).

Game G_4 (Figure 14, boxed statement omitted) is the same as G_3 except that the extraneous y computations in ReLR are removed and, in ReLR, we replace $r_b + r'$ with r' (this change is highlighted in grey). The latter is justified because this r' value is no longer used elsewhere in the game, so $r_b + r'$ is distributed identically to a random value. Thus $\Pr[G_3^{\mathcal{A}} \Rightarrow \mathsf{true}] = \Pr[G_4^{\mathcal{A}} \Rightarrow \mathsf{true}]$.

Game G_5 (Figure 14, boxed statement included) replaces C, τ in ReLR with a random ciphertext. We justify the transition from G_4 to G_5 via a reduction to the one-time, real-or-random security of π_{re}. In more detail, let $\mathcal{C}$ be the OT-ROR$_{\pi_{\mathrm{re}}}$ adversary that works as follows. It runs game G_4 except that it replaces computing the updates of τ_b and C_b in ReLR by a query to its oracle on $(c, (C_b, \tau_b))$ for a counter value c that it increments with each execution of ReLR (the counter ensuring distinct keys are used each time). Adversary $\mathcal{C}$ outputs one if $(b = b')$ after $\mathcal{A}$ finishes. We have by construction that $\Pr[\text{MU-ROR1}^{\mathcal{C}}_{\pi_{\mathrm{re}}} \Rightarrow 1] = \Pr[G_4^{\mathcal{A}} \Rightarrow \mathsf{true}]$. We also have that $\Pr[\text{MU-ROR0}^{\mathcal{C}''}_{\pi_{\mathrm{re}}} \Rightarrow 1] = \Pr[G_5^{\mathcal{A}} \Rightarrow \mathsf{true}]$. Thus,

$$\Pr\left[\, G_4^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] = \Pr\left[\, G_5^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{re}}}(\mathcal{C}) \,.$$

Note that the adversary $\mathcal{C}$ makes at most q queries in total and at most one query on any key index i.

In game G_5 the bit b is not used (once we remove the extraneous code above the boxed statement). Thus $\Pr[G_5^{\mathcal{A}} \Rightarrow \mathsf{true}] = 1/2$. Combining all the above we have the following:

$$\begin{aligned}
\mathsf{Adv}^{\text{up-reenc}}_{\text{ReCrypt},\kappa,t}(\mathcal{A}) &= 2 \cdot \Pr\left[\,\text{UP-REENC}^{\mathcal{A}}_{\text{ReCrypt},\kappa,t} \Rightarrow \mathsf{true}\,\right] - 1 \\
&= 2 \cdot \Pr\left[\,G_0^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] - 1 \\
&\leq 2 \cdot \Pr\left[\,G_1^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\text{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\,G_2^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\text{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\,G_3^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\text{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\,G_4^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\text{kem}},t}(\mathcal{B}) - 1 \\
&\leq 2 \cdot \Pr\left[\,G_5^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\text{kem}},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\text{re}}}(\mathcal{C}) - 1 \\
&= 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\text{kem}},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\text{re}}}(\mathcal{C})\ . \qquad \square
\end{aligned}$$

9.3 Ciphertext Integrity of ReCrypt

Like KSS, ReCrypt protects against ciphertext tampering in two ways. First, the KEM is itself an AE scheme. Second, we include an authenticity tag $\mathcal{H}(m) + F(x, 0)$ within the AE plaintext. The latter should, intuitively, ensure that any tampering with the ciphertext body, which modifies the decrypted message since ciphertext body and message bits are 1-1, are caught. However, we must deal with the fact that UP-INT allows the adversary to obtain the DEM key associated with honest ciphertexts.

For KSS, this was handled by requiring the DEM to be compactly robust. Here the authentication tag $\mathcal{H}(M) + F(x, 0)$ fills this role, and we show that ReCrypt's DEM scheme π_{dem} (as defined at the beginning of Section 9.1) is compactly robust below. First we use it to establish the UP-INT security of ReCrypt, as captured by the following theorem.

Theorem 8 (UP-INT **security of** ReCrypt**).** *Let $\pi_{kem} = (\mathcal{K}_{kem}, \mathcal{E}_{kem}, \mathcal{D}_{kem})$ be an AE scheme, $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a key-homomorphic PRF, $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$ be a hash function, let* ReCrypt *be the scheme as described in Figure 13, and let π_{dem} be the DEM scheme underlying* ReCrypt*. Let $\kappa \geq 0$ and $t > 0$. Then for any* UP-INT$_{\text{ReCrypt},\kappa,t}$ *adversary $\mathcal{A}$ running in time t and making at most q oracle queries with max length σ, there exist adversaries $\mathcal{B}, \mathcal{C}$ such that:*

$$\mathsf{Adv}^{\text{up-int}}_{\text{ReCrypt},\kappa,t}(\mathcal{A}) \leq \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{kem}}(\mathcal{B}) + \mathsf{Adv}^{\text{rob}}_{\pi_{dem}}(\mathcal{C})\ .$$

Adversaries $\mathcal{B}, \mathcal{C}$ run in time that of $\mathcal{A}$ plus a small $\mathcal{O}(q + t + \kappa)$ overhead. Adversary $\mathcal{B}$ makes at most q queries.

The proof is essentially the same as the proof of Theorem 4. We give it for completeness in Appendix B.

Compact robustness analysis. We now turn to showing that the DEM underlying ReCrypt is compactly robust. Trying to do so while assuming only that F is a PRF doesn't work: in the robustness game the keys are chosen by the adversary. We will therefore prove robustness when one instantiates F with the Naor-Pinkas-Reingold key-homomorphic PRF, which is the one we suggest using. Let $(\mathcal{K}, +)$ and $(\mathcal{Y}, +)$ be groups and $H : \{0,1\}^* \to \mathcal{Y}$ be a hash function that we will model as a random oracle. Then $F(x, c) = x \cdot H(c)$. The standard PRF security of this construction rests on the decisional Diffie-Hellman problem being hard in $\mathcal{Y}$ [NPR99].

The robustness analysis of π_{dem} when using this F with a hash function $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$, also modeled as a random oracle, will rely instead on the computational intractability of the discrete log (DL) problem in $\mathcal{Y}$. Formally, we assume that $\mathcal{Y}$ is cyclic and let $P \in \mathcal{Y}$ be some fixed generator. Then $\text{DL}_{\mathcal{Y},P}$ is the game that chooses a random scalar $\beta \in \mathbb{Z}_{|\mathcal{Y}|}$, and runs an adversary $\mathcal{B}$ on input $(P, \beta \cdot P)$. Upon finishing, $\mathcal{B}$ outputs a scalar β' and the game outputs true if $\beta = \beta'$. We associate to $\mathcal{Y}, P$ and $\text{DL}_{\mathcal{Y},P}$-adversary $\mathcal{B}$ the advantage measure

$$\mathsf{Adv}^{\text{dl}}_{\mathcal{Y},P}(\mathcal{B}) = \Pr\left[\,\text{DL}^{\mathcal{B}}_{\mathcal{Y},P} \Rightarrow \mathsf{true}\,\right]\ .$$

Now we can state the following theorem.

Theorem 9. *Let $F(x,c) = x \cdot H(c)$ be the NPR key-homomorphic PRF for hash function $H : \{0,1\}^* \to \mathcal{Y}$ modeled as a random oracle. Let $\mathcal{H} : \{0,1\}^* \to \mathcal{Y}$ be a hash function also modeled as a random oracle, and let π_{dem} be* ReCrypt*'s underlying DEM built from F and $\mathcal{H}$. Then for any $\mathrm{ROB}_{\pi_{dem}}$ adversary $\mathcal{A}$ making at most q random oracle queries, there exists $\mathrm{DL}_{\mathcal{Y},P}$-adversary $\mathcal{B}$ such that:*

$$\mathsf{Adv}^{\mathrm{rob}}_{\pi_{dem}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{dl}}_{\mathcal{Y},P}(\mathcal{B}) + \frac{q^2}{|\mathcal{Y}|} \ .$$

Adversary $\mathcal{B}$ runs in time that of $\mathcal{A}$ plus $\mathcal{O}(q)$ overhead.

Proof. We assume without loss of generality that $\mathcal{A}$ never repeats a query to one of its random oracles. First we replace $\mathcal{H}$ with a randomly chosen injection, call it $\overline{\mathcal{H}}$. Let G be the game that is the same as $\mathrm{ROB}^{\mathcal{A}}_{\pi_{\mathrm{dem}}}$ except that $\mathcal{H}$ is replaced by $\overline{\mathcal{H}}$. A birthday bound argument gives that

$$\Pr\left[\,\mathrm{ROB}^{\mathcal{A}}_{\pi_{\mathrm{dem}}} \Rightarrow \mathsf{true}\,\right] \leq \Pr\left[\,G^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + \frac{q^2}{|\mathcal{Y}|} \ . \tag{4}$$

We now define a $\mathrm{DL}_{\mathcal{Y},P}$ adversary $\mathcal{B}$. Upon being executed with input a DL instance P, Z, adversary $\mathcal{B}$ runs $\mathcal{A}$, responding to $\overline{\mathcal{H}}$ and H queries as follows. For the i^{th} distinct $\overline{\mathcal{H}}(m)$ query, the adversary lets $m_i \leftarrow m$ and chooses α_i at random without replacement from $\mathbb{Z}_{|\mathcal{Y}|}$, meaning all α_i are distinct. It returns $\alpha_i \cdot P$ to $\mathcal{A}$. For the i^{th} distinct $H(c)$ query, the adversary checks if $c = 0$, in which case it returns Z. Otherwise it chooses $\beta_j \xleftarrow{\$} \mathbb{Z}_{|\mathcal{Y}|}$ and returns $\beta_i \cdot P$.

Eventually $\mathcal{A}$ finishes and outputs $\tau, (x,C), (x^*,C^*)$. Let $m = \mathsf{unpad}_{\mathcal{Y}}(C - \hat{F}^{|C|}(x))$ and $m^* = \mathsf{unpad}_{\mathcal{Y}}(C^* - \hat{F}^{|C^*|}(x^*))$. For $\mathcal{A}$ to win, it must be that

(i) $(x,C) \neq (x^*,C^*)$, and
(ii) $\overline{\mathcal{H}}(m) + F(x,0) = \tau = \overline{\mathcal{H}}(m^*) + F(x^*,0)$.

Because $\overline{\mathcal{H}}$ and $\mathsf{unpad}_{\mathcal{Y}}$ are injective, winning therefore requires that $x \neq x^*$. To argue this, consider otherwise, that $x = x^*$. Then necessarily $C \neq C^*$ to satisfy condition (i). But then it must be that $m \neq m^*$ by the injectivity of $\mathsf{unpad}_{\mathcal{Y}}$ and $\overline{\mathcal{H}}$, and plugging into the equations of (ii) implies that the condition cannot hold.

Therefore, if when $\mathcal{A}$ finishes $x = x^*$, the adversary $\mathcal{B}$ can simply output a random value, i.e., give up. Otherwise $\mathcal{B}$ determines the indexes i, j such that $m_i = m$ and $m_j = m^*$. It then sets $\beta' = \frac{\alpha_i - \alpha_j}{x^* - x}$ and outputs β'. If $\mathcal{A}$ wins (4) holds, and so

$$\overline{\mathcal{H}}(m) + F(x,0) = \overline{\mathcal{H}}(m^*) + F(x^*,0) \Leftrightarrow \alpha_i \cdot P + x \cdot \beta \cdot P = \alpha_j \cdot P + x^* \cdot \beta \cdot P \Leftrightarrow \beta = \frac{\alpha_i - \alpha_j}{x^* - x}$$

Thus if $\mathcal{A}$ wins, so does $\mathcal{B}$, and combining with (4) we have justified that

$$\mathsf{Adv}^{\mathrm{rob}}_{\pi_{\mathrm{dem}}}(\mathcal{A}) \leq \Pr\left[\,G^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] + \frac{q^2}{|\mathcal{Y}|} \leq \mathsf{Adv}^{\mathrm{dl}}_{\mathcal{Y},P}(\mathcal{B}) + \frac{q^2}{|\mathcal{Y}|} \ .$$

□

10 Implementation and Performance

This section provides an analysis of the viability of ReCrypt for use in practice.

We first must instantiate the key-homomorphic PRF used by ReCrypt. While BLMR suggest using a standard model construction, a more efficient route is to use the classic ROM construction due originally to Naor, Pinkas, and Reingold [NPR99]. Here one uses $F(k,x) = k \cdot H(x)$ where H is modelled as a random oracle $H : \mathcal{X} \to \mathbb{G}$, $\mathcal{X}$ is any suitable set (bit-strings of some fixed length, for example), and $(\mathbb{G}, +)$ is a group in which the decisional Diffie–Hellman (DDH) assumption holds. We will use as group $\mathbb{G}$ a subset of the points on an elliptic curve $E(\mathbb{F}_p)$ defined over a field of prime order. In our implementation we use the specific curve Curve25519 [Ber06], which has $p = 2^{255} - 19$. We use AES128-GCM for the AE scheme π_{kem} used as the KEM.

We reuse the hash function H, which we relax by assuming elements of $\mathcal{X}$ can be represented by bitstrings, in order to instantiate the random oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$. We separate the domains of H, $\mathcal{H}$ by adding a distinct prefix.

Message encoding. Recall that encryption is done block-wise as $C_i = m_i + F(x,i)$, where m_i are elements of the group $\mathcal{Y}$, here equal to the group $\mathbb{G}$, output by the function $\mathsf{pad}_{\mathcal{Y}}$.

For a padding function, we require the function and its inverse to be efficiently computable. In order to avoid side-channel leakage on plaintexts, we require the function to be amenable to constant-time implementation. In addition, security mandates that $\mathsf{unpad}_{\mathcal{Y}}$ can be implemented in such a way that it rejects on input any element that it is outside the range of $\mathsf{pad}_{\mathcal{Y}}$. For otherwise, $\mathsf{unpad}_{\mathcal{Y}}$ may not act as a bijection on its inputs; the existence of two (sequence of) points P, P' on the curve such that $\mathsf{unpad}_{\mathcal{Y}}(P) = \mathsf{unpad}_{\mathcal{Y}}(P')$ might then allow an adversary to construct a ciphertext forgery.

We construct $\mathsf{pad}_{\mathcal{Y}}, \mathsf{unpad}_{\mathcal{Y}}$ by first applying basic PKCS7 padding [Kal98], to pad the input to a multiple of 31 bytes, and then use an encoding function $\sigma_m : \{0,1\}^{248} \to \mathbb{G}$ to map each block of 31 bytes to a group element.

We use the Elligator function [BHKL13] and its inverse to realize σ_m and σ_m^{-1} respectively. For elliptic curve groups including Curve25519 [Ber06], the Elligator function produces an injective mapping from $\{0, 1, \ldots, \frac{p-1}{2}\}$ to points on the curve. The inverse mapping returns elements of the form $r, p - r$; we obtain a bijection by inverting the map and taking as output the value that is less than $\frac{p-1}{2}$.

Using Elligator, the natural block size for messages would be $\lfloor \log(p-1) - 1 \rfloor = 254$ bits, which is not an integral number of bytes. However, splitting bytes would have a deleterious impact on performance, and so we use a 31 byte (248 bit) block size. Hence, on recovering the message using σ_m^{-1}, we must additionally check that the top byte is zero.

Hashing to the curve. In order to instantiate the key-homomorphic PRF, we require a hash function H outputting elements in the group $\mathbb{G}$. The natural way to build H would be to take a regular hash function $h : \{0,1\}^* \to \{0,1\}^n$, modelled as a random oracle, and apply any encoding function σ_m, resulting in $H(x) := \sigma_m(h(x))$. However, for the function H to additionally act like a random oracle, we would require at least that σ_m maps uniformly to elements in $\mathbb{G}$, imposing additional requirements on σ_m. We therefore require a second encoding function σ_h for constructing the key-homomorphic PRF, and require that σ_h composed with a random oracle h is indifferentiable [MRH04] from a random oracle.

An in-depth treatment of indifferentiable hashing onto elliptic curves appears in [BCI+10]. For simplicity, we opted to use their general approach: $\sigma_h(m) = f(h_1(m)) + h_2(m) \cdot G$. Here h_1 and h_2 are cryptographic hash functions suitable to be modeled as random oracles, instantiated in our implementation by SHA-256 with appropriate domain separation. The function f needs to be what [BCI+10] refer to as a weak encoding function. Elligator satisfies the necessary properties, as it is polynomial time computable, the probability of any point being output is either 0 or $2/\#\mathbb{G}$, and it is samplable by simply picking from $\pm f^{-1}(P)$ for any $P \in E(\mathbb{F}_p)$.

Point compression. Curve25519 supports an x-coordinate-only Montgomery ladder for scalar multiplication. Since we want to unambiguously add points rather than just computing scalar multiples, we need to work with both x and y coordinates. Therefore, we used the twisted Edwards form of Curve25519, implemented in [LdV], which in turn is based on [Lan]. We also used the latter to guide our Elligator implementation.

We save bandwidth using point compression. When a curve point is serialized, only the y coordinate and the sign of the x coordinate (1-bit) needs to be recorded. Since the y coordinate requires less than the full 32 bytes, we are able to serialize points as 32 byte values. Each 32 byte serialized value represents 31 bytes of plaintext, giving a ciphertext expansion of 3%. Upon deserialization, the x coordinate must be recomputed. This requires computing a square root, taking approximately 20 μs. Of course this cost could be avoided by instead serializing both x and y coordinates. This would create a 64 byte ciphertext for each 31 bytes of plaintext, an expansion of 106%. We consider that to be unacceptable.

Microbenchmarks. We built our reference implementation using the Rust [MKI14] programming language. The implementation can be found at [ES]. Our implementation is single-threaded and we measured performance on an Intel CPU (Haswell), running at 3.8GHz.

Figure 15 shows wall clock times for ReCrypt operations over various plaintext sizes. As might be expected given the nature of the cryptographic operations involved, performance is far from competitive with conventional AE schemes. For comparison, AES-GCM on the same hardware platform encrypts 1 block, 1 KB, 1 MB and 1 GB of plaintext in 15 μs, 24 μs, 9 ms, and 11 s, respectively. KSS has performance determined by that of AES-GCM, while the performance of the ReCrypt scheme is largely determined by the scalar multiplications required to evaluate the PRF. Independent of message length there is a 1000x performance cost to achieve our strongest notion of security.

ReCrypt Operation	Time per CPU 1 block	1 KB	1 MB	1 GB	cycles/byte
Encrypt	510 μs	8.7 ms	8.5 s	2.5 hours	32.3 K
ReEnc	241 μs	6.7 ms	7.3 s	2.0 hours	27.7 K
Decrypt	536 μs	7.3 ms	7.6 s	2.2 hours	28.9 K
ReKeyGen (total)		220 μs			836 K

Fig. 15. Processing times for ReCrypt operations measured on a 3.8GHz CPU. 1 block represents any plaintext $\leq$ 31 bytes. Number of iterations: 1000 (for 1 block, 1 KB), 100 (for 1 MB) and 1 (for 1 GB). Cycles per byte given for 1MB ciphertexts.

Discussion. Given the performance difference, ReCrypt is best suited to very small or very valuable plaintexts (ideally, both). If the plaintext corpus is moderately or very large, cost and performance may prohibit practitioners from using ReCrypt over more performant schemes like KSS that give strictly weaker security. To make this discussion concrete, we consider two examples in more detail.

For privacy and security reasons, regulation mandates that credit card numbers must be stored in encrypted form [PCI16]. These include the recommendation that a mechanism must be in place to rotate keys on a regular basis and in the face of known or suspected compromise. Given the sensitive nature of payment information, the naive solution of decrypting and encrypting the data to rotate keys exposes it to some risk, since it makes the data available in plaintext form for at least a period of time. Similarly, NIST guidelines [Bar16] recommend balancing the risk introduced by the re-encryption process with the benefits offered by key rotation. On the other hand, our updatable AE schemes enable secure rotation of keys using an untrusted storage service.

Consider a payment system with, say, 1 billion credit card entries. The storage required for encrypted credit card numbers using ReCrypt is about 30 GB (assuming one block per entry). Projecting from performance measurements in Figure 15, a full key rotation across the this dataset requires 60 CPU-hours: a significant amount of computation, but liekly acceptable given infrequent rotations and many available CPUs. The time to decrypt a single entry is sub-millisecond; this is small compared to processing time for a credit card transaction.

As a second example we consider long-term storage of static data, commonly known as "deep" or "cold" storage. Such data are accessed infrequently, yet data owners may still desire (or be required to) periodically rotate the encryption keys used for protecting the data. In such cases it may be more convenient for the data owner to allow the storage provider to rotate the encryption keys using a system local to the data, as opposed to the data owner retrieving the data and performing the re-encryption.

For a rough estimation of costs, we compute the cost of performing re-encryption using ReCrypt on Amazon Web Service's Elastic Compute Cloud (AWS EC2). Using the price per CPU-hour of an AWS EC2 instance of 0.05 \$/hour (USD)[9], we compute the cost to perform updates using ReCrypt as 0.10 \$/GB. For small- to medium-sized data sets or for data sets that are particularly valuable (e.g. financial information), this cost may be justified. However, for moderately-sized to large data sets, the cost may be prohibitive and clients may favor basic security schemes like KSS.

Acknowledgements

We thank the anonymous Crypto reviewers for their feedback and discussion relating to a bug in our initial KSS construction, and for their additional comments and suggestions. We thank Joseph Jaeger and Anja Lehmann for pointing out bugs in previous versions of the paper and for many discussions about definitions and our analyses. We thank Michael Klooss for feedback relating to the proofs of security. This work was supported in part by NSF grant CNS-1330308, EPSRC grants EP/K035584/1 and EP/M013472/1, by a research programme funded by Huawei Technologies and delivered through the Institute for Cyber Security Innovation at Royal Holloway, University of London, and a gift from Microsoft.

[9] This rate is based on the lowest per-CPU cost on AWS as of June 2017. Namely, an m4.16xlarge instance available in the AWS US-East (Ohio) region which provides 64 CPUs and is available on-demand for 3.20 \$/hour.

References

ABBC10. Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash. Cryptographic agility and its relation to circular encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 403–422. Springer, Heidelberg, May 2010.

ABL+14. Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125. Springer, Heidelberg, December 2014.

ABN10. Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 480–497. Springer, Heidelberg, February 2010.

AWS. AWS. Protecting data using client-side encryption. `http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingClientSideEncryption.html`.

Bar16. Elaine Barker. Recommendation for key management Part 1: General. In *NIST special publication 800-57 Part 1 Revision 4*, 2016.

BCI+10. Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.

BDPS14. Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. On symmetric encryption with distinguishable decryption failures. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 367–390. Springer, Heidelberg, March 2014.

Ber06. Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

BHKL13. Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 967–980. ACM Press, November 2013.

BLMR13. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.

BLMR15. Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. Cryptology ePrint Archive, Report 2015/220, 2015. `http://eprint.iacr.org/2015/220`.

BN00. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, December 2000.

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Vaudenay [Vau06], pages 409–426.

BRS03. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, Heidelberg, August 2003.

CGH12. David Cash, Matthew Green, and Susan Hohenberger. New definitions and separations for circular security. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 540–557. Springer, Heidelberg, May 2012.

CH07. Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 185–194. ACM Press, October 2007.

CK05. DL Cool and Angelos D Keromytis. Conversion and proxy functions for symmetric key ciphers. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 1, pages 662–667. IEEE, 2005.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

DGRW18. Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In *Advances in Cryptology – CRYPTO*, 2018.

ES. Adam Everspaugh and Sam Scott. ReCrypt prototype implementation. `https://github.com/samscott89/recrypt`.

FLPQ13. Pooya Farshim, Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust encryption, revisited. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 352–368. Springer, Heidelberg, February / March 2013.

FOR17. Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symm. Cryptol.*, 2017(1):449–473, 2017.

GLR17. Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017.

Goo. Google. Managing data encryption. `https://cloud.google.com/storage/docs/encryption`.

ID03. Anca Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *NDSS 2003*. The Internet Society, February 2003.

Kal98. Burt Kaliski. Pkcs #7: Cryptographic message syntax version 1.5. RFC 2315, RFC Editor, March 1998. `http://www.rfc-editor.org/rfc/rfc2315.txt`.

KRW15. Venkata Koppula, Kim Ramchen, and Brent Waters. Separations in circular security for arbitrary length key cycles. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 378–400. Springer, Heidelberg, March 2015.

Lan. Adam Langley. ed25519 for go. `http://godoc.org/github.com/agl/ed25519`.

LdV. Isis Agora Lovecruft and Henry de Valence. curve25519-dalek. `https://github.com/dalek-cryptography/curve25519-dalek/`.

MKI14. Nicholas D Matsakis and Felix S Klock II. The rust language. *ACM SIGAda Ada Letters*, 34(3):103–104, 2014.

MRH04. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

NPR99. Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Heidelberg, May 1999.

NRS14. Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014.

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

PRS11. Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 372–389. Springer, Heidelberg, December 2011.

Rog04. Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004.

RS06. Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Vaudenay [Vau06], pages 373–390.

RSS11. Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Heidelberg, May 2011.

Vau06. Serge Vaudenay, editor. *EUROCRYPT 2006*, volume 4004 of *LNCS*. Springer, Heidelberg, May / June 2006.

WC81. Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

A Proof of Theorem 6 (UP-IND security of ReCrypt)

As before, we write Π to denote the ReCrypt scheme. Our proof uses a sequence of games. $\eth G_0$ (Figure 16, boxed statements omitted) is the UP-IND$_{\Pi,\kappa,t}$ game. In this game, and in all subsequent ones, we omit the main procedure to save space. That procedure picks a random bit b, generates secret keys $k_1, \ldots, k_{t+\kappa} \leftarrow\!\!\$\ \mathcal{K}_{\text{kem}}$, runs $\mathcal{A}$ with input $k_{t+1}, \ldots, k_{t+\kappa}$ and with access to the oracles, and returns `true` if $\mathcal{A}$'s output b' is equal to b.

Game G_1 (Figure 16, boxed statements included) replaces the KEM encryption when used with honest keys with randomly chosen ciphertexts and table look-ups to implement decryption. To bound the transition between G_0 and G_1 we use a reduction to the AE security of the KEM. In detail, let $\mathcal{B}$ be the MU-ROR-AE$_{\pi,t}$ adversary that works exactly like game G_0 (and G_1) except that: (1) uses of $\mathcal{E}_{\text{kem}}(k_i, m)$ with $i \le t$ are replaced with a query to $\mathcal{B}$'s encryption oracle on (i, m) and (2) uses of $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$ with $i \le t$ are replaced

with a query to $\mathcal{B}$'s decryption oracle on $(i, \tilde{C})$. By construction, it holds that $\text{MU-ROR-AE0}^{\mathcal{B}}_{\pi,t} = G_0^{\mathcal{A}}$ and $\text{MU-ROR-AE1}^{\mathcal{B}}_{\pi,t} = G_1^{\mathcal{A}}$. Thus,

$$\Pr\left[G_0 \Rightarrow \mathsf{true}\right] \leq \Pr\left[G_1 \Rightarrow \mathsf{true}\right] + \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi,t}(\mathcal{B}) \ .$$

We now will make transitions so that the transcript observed by the adversary is independent of the challenge bit b. This will involve: (1) showing that headers returned in response to invalid ReEnc queries are indistinguishable from encryptions of a random DEM key and the tag (because of the secret-sharing of DEM keys and refreshing of them during re-encryption); (2) the y values generated during challenge queries are never revealed to the adversary (which we argue by induction using the rules preventing certain re-key or re-encryption queries); and (3) applying the OT-ROR security of π_{dem} to argue that the ciphertext body leaks nothing about the challenge bit b.

Towards this we start by modifying how ReEnc returns headers in the case of an invalid query (one for which $\mathtt{T}[i, \tilde{C}] \neq \perp$ and $j > t$). Notice that in game G_1, in the case that just the header is returned in response to a ReEnc query, the header's plaintext portion $y + x' + r'$ is independent of $y + x'$ because r' is uniform and not used elsewhere in the game. We make this explicit in game G_2 (Figure 16, boxed statement omitted) which has ReEnc return a $\mathcal{E}_{\text{kem}}(k_j, r' \| \tau')$ instead of $\mathcal{E}_{\text{kem}}(k_j, (y + x' + r') \| \tau')$. Game G_2 also dispenses with the now redundant lines of code from G_1's handling of E_{kem}, D_{kem}. We have that $\Pr[G_1^{\mathcal{A}} \Rightarrow \mathsf{true}] = \Pr[G_2^{\mathcal{A}} \Rightarrow \mathsf{true}]$. Note that this step of the proof relies on the secret sharing of the DEM key — without it we would not be able to argue independence and, in fact, the returned ciphertext header would reveal the DEM key.

In game G_3, we replace the DEM key share y in LR with a constant value. We will argue that

$$\Pr\left[\, G_2^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] = \Pr\left[\, G_3^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] \ , \tag{5}$$

by arguing that no values returned to the adversary depend on y. Intuitively, the reason is that while y may affect other entries of $\mathtt{C}$, via re-key or re-encryption queries, the rules about invalid queries combined with our earlier game transition to G_2 ensure that these plaintext values never impact any values returned to the adversary.

To argue this, consider any query to LR on index i^* that produces an entry $\mathtt{C}[i^*, \tilde{C}^*] = y \| \tau$. Necessarily $\mathtt{T}[i^*, \tilde{C}^*] \neq \perp$. We perform an inductive argument over the subsequent queries to show the following condition holds after each query: the only values in the game dependent on y are entries in $\mathtt{C}[j, \tilde{C}]$ for some $j, \tilde{C}$ for which $\mathtt{T}[j, \tilde{C}] \neq \perp$. In particular, no values returned to the adversary are dependent. By dependent we mean that the values are a function of y and (possibly) other variables defined in the game. Below we refer to entries $j, \tilde{C}$ for which $\mathtt{C}[j, \tilde{C}]$ depends on y as dependent entries.

As base case consider that the LR query was the last query. Then the condition holds by inspection of the code of LR in game G_2. Now consider an arbitrary later query, and assume that the condition holds at the end of the previous query. We have the following cases broken down by the type of query.

(1) The query is to Enc. Then no dependent entries are accessed, and so the condition trivially holds after the query.

(2) The query is to ReKeyGen on a triple $i, j, \tilde{C}$. If $i, \tilde{C}$ is not a dependent entry, then the condition trivially holds after the query. Otherwise, by the inductive hypothesis $\mathtt{T}[i, \tilde{C}] \neq \perp$ and so for $\mathtt{C}[i, \tilde{C}]$ to be accessed, necessarily $\mathsf{Hon}(j)$. Then the procedure generates a new dependent plaintext submitted to E_{kem} for $\mathsf{Hon}(j)$, thereby adding a new dependent entry $\mathtt{C}[j, \tilde{C}']$. Because $\mathtt{T}[i, \tilde{C}] \neq \perp$ it holds that $\mathtt{T}[j, \tilde{C}']$ is set to a value other than $\perp$. None of the returned values depend on either dependent entry. Thus the condition holds after the query completes.

(3) The query is to ReEnc on a triple $i, j, (\tilde{C}, (r, C))$. If $i, \tilde{C}$ is not a dependent entry, then the condition trivially holds after the query. Otherwise, by the inductive hypothesis $\mathtt{T}[i, \tilde{C}] \neq \perp$. If $\mathsf{Mal}(j)$ then no new dependent entries are generated (the KEM plaintext in this case using r' instead of $y \oplus r'$). If $\mathsf{Hon}(j)$, then the procedure generates a new dependent plaintext submitted to E_{kem} for $\mathsf{Hon}(j)$, adding a new dependent entry $\mathtt{C}[j, \tilde{C}']$. Because $\mathtt{T}[i, \tilde{C}] \neq \perp$ it holds that $\mathtt{T}[j, \tilde{C}']$ is set to a value other than $\perp$. None of the returned values depend on either dependent entry. Thus the condition holds after the query completes.

Thus by induction the condition holds for all queries, and we have justified (5).

Game G_4 (Figure 16, boxed statement omitted) is the same as G_3 except that the extraneous y computations in LR are removed. Game G_5 (Figure 16, boxed statement included) replaces C, τ in LR with

$\underline{\text{Enc}(i,m)}$ G_0 , $\boxed{G_1}$
$x, r \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{\ell(m)}(x)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\!\$\ \text{E}_{\text{kem}}(i, (x + r) \parallel \tau)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\bot$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $(r, C) \leftarrow \mathtt{T}[i, \tilde{C}]$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C + \hat{F}^{|C|}(x'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
$C' \leftarrow C + \hat{F}^{|C|}(x')$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\tilde{C}'$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C')$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\text{LR}(i, m_0, m_1)}$
If $\mathsf{Mal}(i)$ or $|m_0| \neq |m_1|$ then Return $\bot$
$x, r \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau \leftarrow \mathcal{H}(m_b) + F(x, 0)$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m_b) + \hat{F}^{\ell(m_0)}(x)$
$y \leftarrow x + r$
$\tilde{C} \leftarrow\!\!\$\ \text{E}_{\text{kem}}(i, y \parallel \tau)$
$\mathtt{T}[i, \tilde{C}] \leftarrow (r, C)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{E}_{\text{kem}}(i, m)}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{E}_{\text{kem}}(k_i, m)$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{E}_{\text{kem}}(k_i, m)$
$\boxed{\tilde{C} \leftarrow\!\!\$\ \mathcal{CS}_{\pi_{\text{kem}}}(|m|)\ ;\ \mathtt{C}[i, \tilde{C}] \leftarrow m}$
Return $\tilde{C}$

$\underline{\text{D}_{\text{kem}}(i, \tilde{C})}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$m \leftarrow \mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$\boxed{m \leftarrow \mathtt{C}[i, \tilde{C}]}$
Return m

$\underline{\text{Enc}(i,m)}$ G_3 , $\boxed{G_4}$
$x, r \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{\ell(\mathsf{pad}_{\mathcal{Y}}(m))}(x)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\!\$\ \text{E}_{\text{kem}}(i, (x + r) \parallel \tau)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\bot$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $(r, C) \leftarrow \mathtt{T}[i, \tilde{C}]$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C + \hat{F}^{|C|}(x'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\tau' \leftarrow\!\!\$\ \tau + F(x', 0)$
  $\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, r' \parallel \tau')$
  Return $\tilde{C}'$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
$C' \leftarrow C + \hat{F}^{|C|}(x')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C')$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\text{LR}(i, m_0, m_1)}$
If $\mathsf{Mal}(i)$ or $|m_0| \neq |m_1|$ then Return $\bot$
$x, r \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau \leftarrow \mathcal{H}(m_b) + F(x, 0)$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m_b) + \hat{F}^{\ell(m_0)}(x)$
$y \leftarrow x + r$ $\boxed{\ ;\ y \leftarrow 0^{|r|}}$
$\tilde{C} \leftarrow\!\!\$\ \text{E}_{\text{kem}}(i, y \parallel \tau)$
$\mathtt{T}[i, \tilde{C}] \leftarrow (r, C)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{E}_{\text{kem}}(i, m)}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{E}_{\text{kem}}(k_i, m)$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{CS}_{\pi_{\text{kem}}}(|m|)\ ;\ \mathtt{C}[i, \tilde{C}] \leftarrow m$
Return $\tilde{C}$

$\underline{\text{D}_{\text{kem}}(i, \tilde{C})}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$m \leftarrow \mathtt{C}[i, \tilde{C}]$
Return m

$\underline{\text{Enc}(i,m)}$ G_5 , $\boxed{G_6}$
$x, r \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m) + \hat{F}^{\ell(m)}(x)$
$\tau \leftarrow \mathcal{H}(m) + F(x, 0)$
$\tilde{C} \leftarrow\!\!\$\ \text{E}_{\text{kem}}(i, (x + r) \parallel \tau)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{ReKeyGen}(i, j, \tilde{C})}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then Return $\bot$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau' \leftarrow \tau + F(x', 0)$
$\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $(r, C) \leftarrow \mathtt{T}[i, \tilde{C}]$
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C + \hat{F}^{|C|}(x'))$
Return $(\tilde{C}', (x', r'))$

$\underline{\text{ReEnc}(i, j, (\tilde{C}, (r, C)))}$
$(y \parallel \tau) \leftarrow \text{D}_{\text{kem}}(i, \tilde{C})$
If $(y \parallel \tau) = \bot$ then Return $\bot$
$x', r' \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
If $\mathsf{Mal}(j) \wedge \mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\tau' \leftarrow\!\!\$\ \mathcal{Y}$; $\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, r' \parallel \tau')$
  Return $\tilde{C}'$
$\tau' \leftarrow \tau + F(x', 0)$
$C' \leftarrow C + \hat{F}^{|C|}(x')$
$\tilde{C}' \leftarrow\!\!\$\ \text{E}_{\text{kem}}(j, (y + x' + r') \parallel \tau')$
If $\mathtt{T}[i, \tilde{C}] \neq \bot$ then
  $\mathtt{T}[j, \tilde{C}'] \leftarrow (r + r', C')$
Return $(\tilde{C}', (r + r', C'))$

$\underline{\text{LR}(i, m_0, m_1)}$
If $\mathsf{Mal}(i)$ or $|m_0| \neq |m_1|$ then Return $\bot$
$x, r \leftarrow\!\!\$\ \mathcal{K}_{\text{dem}}$
$\tau \leftarrow \mathcal{H}(m_b) + F(x, 0)$
$C \leftarrow \mathsf{pad}_{\mathcal{Y}}(m_b) + \hat{F}^{\ell(m_0)}(x)$
$\boxed{C \leftarrow\!\!\$\ \mathcal{Y}^{|\mathsf{pad}_{\mathcal{Y}}(m_0)|}\ ;\ \tau \leftarrow\!\!\$\ \mathcal{Y}}$
$\tilde{C} \leftarrow\!\!\$\ \text{E}_{\text{kem}}(i, 0^{|r|} \parallel \tau)$
$\mathtt{T}[i, \tilde{C}] \leftarrow (r, C)$
Return $(\tilde{C}, (r, C))$

$\underline{\text{E}_{\text{kem}}(i, m)}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{E}_{\text{kem}}(k_i, m)$
$\tilde{C} \leftarrow\!\!\$\ \mathcal{CS}_{\pi_{\text{kem}}}(|m|)\ ;\ \mathtt{C}[i, \tilde{C}] \leftarrow m$
Return $\tilde{C}$

$\underline{\text{D}_{\text{kem}}(i, \tilde{C})}$
If $\mathsf{Mal}(i)$ then Return $\mathcal{D}_{\text{kem}}(k_i, \tilde{C})$
$m \leftarrow \mathtt{C}[i, \tilde{C}]$
Return m

Fig. 16. Games for proof of UP-IND for ReCrypt.

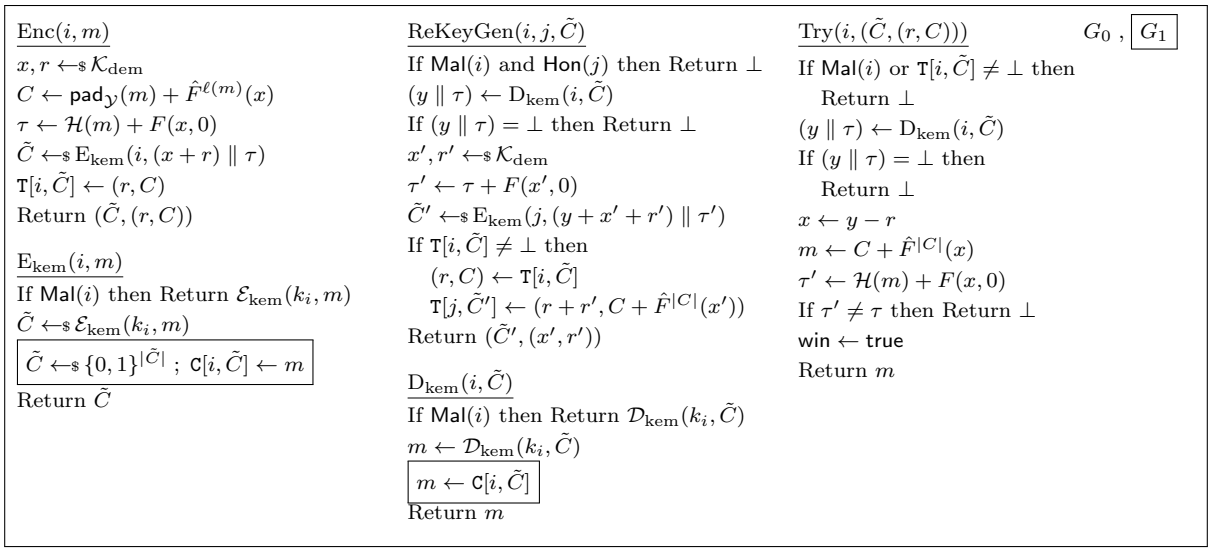

Fig. 17. Games for proof of UP-INT for ReCrypt.

a random ciphertext. We justify the transition from G_4 to G_5 via a reduction to the one-time, real-or-random security of π_{dem}. In more detail, let $\mathcal{C}$ be the OT-ROR$_{\pi_{\mathrm{dem}}}$ adversary that works as follows. It runs game G_4 except that it replaces computing the DEM encryption in LR by a query to its oracle on (c, m_b) for a counter value c that it increments with each execution of LR (the counter ensuring distinct keys are used each time). Adversary $\mathcal{C}$ outputs one if $(b = b')$ after $\mathcal{A}$ finishes. We have by construction that $\Pr[\text{MU-ROR1}^{\mathcal{C}}_{\pi_{\mathrm{dem}}} \Rightarrow 1] = \Pr[G_4^{\mathcal{A}} \Rightarrow \mathsf{true}]$. We also have that $\Pr[\text{MU-ROR0}^{\mathcal{C}''}_{\pi_{\mathrm{dem}}} \Rightarrow 1] = \Pr[G_5^{\mathcal{A}} \Rightarrow \mathsf{true}]$. Thus,

$$\Pr\left[\, G_4^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] = \Pr\left[\, G_5^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{dem}}}(\mathcal{C}) \; .$$

Note that the adversary $\mathcal{C}$ makes at most q queries in total and at most one query on any key index i.

In game G_5 the bit b is not used (once we remove the extraneous code above the boxed statement). Thus $\Pr[G_5^{\mathcal{A}} \Rightarrow \mathsf{true}] = 1/2$. Combining all the above we have the following:

$$\begin{aligned}
\mathsf{Adv}^{\text{up-ind}}_{\mathrm{ReCrypt},\kappa,t}(\mathcal{A}) &= 2 \cdot \Pr\left[\, \text{UP-IND}^{\mathcal{A}}_{\mathrm{ReCrypt},\kappa,t} \Rightarrow \mathsf{true} \,\right] - 1 \\
&= 2 \cdot \Pr\left[\, G_0^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] - 1 \\
&\leq 2 \cdot \Pr\left[\, G_1^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\, G_2^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\, G_3^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&= 2 \cdot \Pr\left[\, G_4^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) - 1 \\
&\leq 2 \cdot \Pr\left[\, G_5^{\mathcal{A}} \Rightarrow \mathsf{true} \,\right] + 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{dem}}}(\mathcal{C}) - 1 \\
&= 2 \cdot \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi_{\mathrm{kem}},t}(\mathcal{B}) + 2 \cdot \mathsf{Adv}^{\text{ot-ror}}_{\pi_{\mathrm{dem}}}(\mathcal{C}) \; .
\end{aligned}$$

□

B Proof of Theorem 8 (UP-INT Security of ReCrypt)

As before, we write Π to denote the ReCrypt scheme. Our proof uses a sequence of two games. Game G_0 (Figure 17, boxed statements omitted) implements $\text{UP-INT}^{\mathcal{A}}_{\Pi,\kappa,t}$. In both games we omit the main procedure. That procedure generates secret keys $k_1, \ldots, k_{t+\kappa} \leftarrow\!\!\$\ \mathcal{K}_{\mathrm{kem}}$, runs $\mathcal{A}$ with input $k_{t+1}, \ldots, k_{t+\kappa}$ and with access to the oracles, and returns true if $\mathsf{win} = \mathsf{true}$ after $\mathcal{A}$ finishes executing.

Game G_1 (Figure 17, boxed statements included) replaces the KEM encryption when used with honest keys with randomly chosen ciphertexts and table look-ups to implement decryption. We bound this transition with the usual reduction to an adversary $\mathcal{B}$. We omit the details justifying that

$$\Pr[\, G_0 \Rightarrow \mathsf{true}\,] \leq \Pr[\, G_1 \Rightarrow \mathsf{true}\,] + \mathsf{Adv}^{\text{mu-ror-ae}}_{\pi,t}(\mathcal{B})\ .$$

Let us refer to the first query to Try that sets win to true as a winning query. It is without loss to assume that no query follows a winning query, making it the winning query. Consider the winning query and let its input be $i^*, (\tilde{C}^*, (r^*, C^*))$. We will show that there exists another query that, together with the winning query, defines values $\tau^*, (x, C), (x^*, C^*)$ that suffice for winning game $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$. We will therefore be able to define a reduction implying that $\mathcal{A}$ winning in game G_1 implies violating the compact robustness of π_{dem}.

Towards this, note that in game G_1 it must be that the winning query accesses $\mathtt{C}[i^*, \tilde{C}^*]$ because i^* must be an honest key. Let $x^*, y^*, r^*, C^*, m^*, \tau^*$ be the variables in Try associated to the winning query, with $\mathtt{C}[i^*, \tilde{C}^*] = (r^*, C^*)$. We have that $\mathcal{D}_{\mathrm{dem}}(x^*, (C^*, \tau^*)) \neq \bot$ because it was a winning query. Since $\mathtt{C}[i^*, \tilde{C}^*] \neq \bot$, there exists a previous query $\mathrm{Enc}(i^*, m)$ or $\mathrm{ReKeyGen}(i, i^*, \tilde{C})$ that set $\mathtt{C}[i^*, \tilde{C}^*]$.

First consider the former case, that the previous query was to $\mathrm{Enc}(i^*, m)$. Necessarily this generated a valid ciphertext (C, τ) and random value r such that $\mathtt{T}[i^*, \tilde{C}^*] = (r, C)$. It must be that $\tau = \tau^*$ and $x + r = y^*$ because $\mathtt{C}[i^*, \tilde{C}^*] = (y^* \| \tau^*)$. Moreover, because $\mathtt{T}[i^*, \tilde{C}^*] \neq (r^*, C^*)$ for a winning query, it must be that $(r, C) \neq (r^*, C^*)$. Thus either $x \neq x^*$ or $C \neq C^*$. We therefore let our $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$ winning values be $\tau^*, (x, C), (x^*, C^*)$.

Second consider the other case, that the previous query was to $\mathrm{ReKeyGen}(i, i^*, \tilde{C})$. Assume for the moment that this query set $\mathtt{T}[i^*, \tilde{C}^*] = (r + r', C)$. Let r, r', x', y, C, τ be the values defined in handling the re-key generation query. For the Try query to be winning, then, it must be that $(r + r', C) \neq (r^*, C^*)$. Because this re-key query set $\mathtt{C}[i^*, \tilde{C}^*] = y^* \| \tau^* = y \| \tau$ we have that $\tau = \tau^*$ and that

$$x + x' + (r + r') = y = y^* = x^* + r^*$$

where we let $x = y - (r + r' + x')$. Combining the above we have that $(x + x', C) \neq (x^*, C^*)$ and so we can set the $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$ winning values to be $\tau^*, (x + x', C), (x^*, C^*)$.

What remains is to argue that $\mathtt{T}[i^*, \tilde{C}^*]$ was indeed necessarily set in this query. We do so by proving that for $\mathrm{ReKeyGen}(i, i^*, \tilde{C})$ to set $\mathtt{C}[i^*, \tilde{C}^*]$ there must have previously been a query $(\tilde{C}_0, (r_0, C_0)) \leftarrow\!\!\$\ \mathrm{Enc}(i_0, m)$ followed by $\gamma \geq 1$ queries:

$$(\tilde{C}_1, x_1, r_1) \leftarrow\!\!\$\ \mathrm{ReKeyGen}(i_0, i_1, \tilde{C}_0)\ ,\ \ldots\ ,\ (\tilde{C}^*, x', r') \leftarrow\!\!\$\ \mathrm{ReKeyGen}(i_{\gamma-1}, i^*, \tilde{C}_{\gamma-1})\ .$$

where we've let $i_{\gamma-1} = i$ and $\tilde{C}_{\gamma-1} = \tilde{C}$. The last query represents the rekey generation query that set $\mathtt{C}[i^*, \tilde{C}^*]$. For the latter to have occurred, it must be that key $i_{\gamma-1}$ is honest otherwise the query would have immediately returned $\bot$ (because i^* is also honest). A previous query must therefore have set $\mathtt{C}[i_{\gamma-1}, \tilde{C}_{\gamma-1}]$. If that query was a ReKeyGen query, then repeat the above reasoning until one arrives at the $(\tilde{C}_0, (r_0, C_0)) \leftarrow\!\!\$\ \mathrm{Enc}(i_0, \tilde{C}_0)$ query. Now in this query it is the case that $\mathtt{T}[i_0, \tilde{C}_0]$ was necessarily set to a value other than $\bot$. In turn, the next $\mathrm{ReKeyGen}(i_0, i_1, \tilde{C}_0)$ query had $\mathtt{T}[i_0, \tilde{C}_0] \neq \bot$, and therefore set $\mathtt{T}[i_1, \tilde{C}_1]$ to a value, and so on until we have that $\mathtt{T}[i^*, \tilde{C}^*]$ is set to something other than $\bot$.

In all cases we have shown that a winning query leads to a $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$ winning set of values. Let $\mathcal{C}$ be the $\mathrm{ROB}_{\pi_{\mathrm{dem}}}$-adversary that runs G_1 and, upon a winning query, determines the previous ReKeyGen query and generates the appropriate winning pair. This can be done in time sub-linear in the number of queries by searching over entries in $\mathtt{C}$. By our arguments above, we have that:

$$\Pr\left[\, G_1^{\mathcal{A}} \Rightarrow \mathsf{true}\,\right] \leq \mathsf{Adv}^{\mathrm{rob}}_{\pi_{\mathrm{dem}}}(\mathcal{C})\ .$$