Towards Leakage-Resilient Ratcheted Key Exchange

Daniel Collins[1], Simone Colombo[2], and Sina Schaeffler[3,4]

[1] Texas A&M University
[2] King's College London, London, UK
[3] ETH Zurich, Zurich, Switzerland
[4] IBM Research Zurich, Switzerland
danielpatcollins@gmail.com, simone.colombo@kcl.ac.uk, sschaeffle@ethz.ch

Abstract. Ratcheted key exchange (RKE) is at the heart of modern secure messaging, enabling protocol participants to continuously update their secret material to protect against *full* state exposure through forward security (protecting past secrets and messages) and post-compromise security (recovering from compromise). However, many practical attacks only provide the adversary with *partial* access to a party's secret state, an attack vector studied under the umbrella of *leakage resilience*. Existing models of RKE provide suboptimal guarantees under partial leakage due to inherent limitations in security under full state exposure.
In this work, we initiate the study of *leakage-resilient ratcheted key exchange* that provides typical guarantees under full state exposure and additional guarantees under partial state exposure between ratchets of the protocol. We consider *unidirectional* ratcheted key exchange (URKE) where one party acts as the sender and the other as receiver. Building on the notions introduced by Balli, Rösler and Vaudenay (ASIACRYPT 2020), we formalise a key indistinguishability game under randomness manipulation and *bounded leakage* (KIND), which in particular enables the adversary to continually leak a bounded amount of the sender's state between honest send calls. We construct a corresponding protocol from a key-updatable key encapsulation mechanism (kuKEM) and a leakage-resilient one-time MAC. By instantiating this MAC in the random oracle model (ROM), results from Balli, Rösler and Vaudenay imply that in the ROM, kuKEM and KIND-secure URKE are equivalent, i.e., can be built from each other. To address the strong limitations that key indistinguishability imposes on the adversary, we formalise a one-wayness game that also permits leakage on the receiver. We then propose a corresponding construction from leakage-resilient kuKEM, which we introduce, and a leakage-resilient one-time MAC. We further show that leakage-resilient kuKEM and one-way-secure URKE are equivalent in the ROM, highlighting the cost that strong one-way security entails. Our work opens exciting directions for developing leakage-resilient messaging protocols.

Keywords: Leakage resilience · Ratcheted key exchange · Secure messaging · Forward security · Post-compromise security

Authors are listed in alphabetical order. Work partially done while Collins, Colombo and Schaeffler were at EPFL, as well as while Collins was at Purdue University and Georgia Institute of Technology.

1 Introduction

In the last decade, the security of messaging solutions has drastically increased. Thanks to the work of researchers and practitioners alike, much of our communications are today protected by strong security guarantees such as forward security, which ensures confidentiality of messages sent before a state exposure [BG21], and post-compromise security, which enables automatic healing after a compromise [CCG16]. In practice, Signal's widely deployed Double Ratchet protocol [PM16; EM19] provides these guarantees, which have been formally captured in the literature by primitives like *ratcheted key exchange* [Bel+17], where shared keys between parties are continually generated over time.

Existing models for messaging schemes using *ratcheting* assume that the adversary gains access to the participants' *full* state, covering scenarios like physical device access [CCG16] or malware infection, such as Pegasus [Sco+22]. In practice, there are also attacks, that only provide the adversary with *partial* information about the secret state of a given party or set of parties, for example many side-channel attacks [GMO01; ADW09b; Fan+10; Spr+18]. Nonetheless, there are stronger limitations to the security one can achieve under full state exposure, than under partial state exposure. This causes limitations in the ability of existing models to capture partial leakage. First, the requirement for the protocol to be correct limits security: for example, exposing a party while their counterpart has sent them ciphertexts that are in transit must allow the adversary to decrypt them. Second, since protecting against the largest set of possible attacks, i.e., achieving some notion of *optimal* security, is inherently expensive [JS18; PR18; BRV20a; Alw+20b], many protocols, such as the Double Ratchet, provide only sub-optimal security guarantees for the sake of performance.

The field of *leakage-resilient* cryptography [DP08] has dealt with the problem of ensuring security in the presence of partial leakage. This therefore raises a natural research question:

Can we design a messaging scheme that provides security guarantees under full state exposure and even stronger guarantees under partial state exposure?

In this paper, we make progress towards answering the above question in the affirmative. We augment the security notions for two-party ratcheted key exchange [Bel+17] with security guarantees under partial state exposure. Ratcheted key exchange, in which a sequence of keys are established over time, captures the core of modern secure messaging. In this work, we focus on *unidirectional* ratcheting (URKE), where the communication flow between two participants goes in a single direction: Alice can only send and Bob can only receive. This approach was first taken in the seminal work of Bellare et al. [Bel+17], and allows for a systematic and modular exploration of the complexity of ratcheting [PR18; BRV20a]. It has also been taken to explore new security guarantees for ratcheting, notably anonymity under state exposure [Dow+22].

The classic leakage model captures *bounded leakage* [Dzi06], where the adversary can adaptively leak arbitrary functions of a secret, provided that the output is limited to an a priori bounded number of bits. We thus take advantage of the fact that keys are continually updated in RKE by enabling the adversary to leak a bounded number of bits between each "round" of protocol execution, i.e., between two consistent send and receive calls. This approach enables us to provide security guarantees under the *continual leakage* of secret material, without having to rely on fixed (public) keying material, which is less general and seems to lead to more expensive constructions, as seen in settings like forward security under continual leakage [Bel+17; Cha+23].

1.1 Summary of contributions

In this paper, we initiate the study of ratcheted key exchange that provides strong security guarantees under *partial* state exposure and make the following contributions towards understanding its complexity along the way:

- In Section 3, we define the notion of one-way security under randomness manipulation and bounded leakage on the receiver (KUOWR) for a key-updatable key encapsulation mechanism (kuKEM) [PR18; BRV20a].
- In Section 4, we introduce the notion of leakage-resilient key indistingushability (LR-KIND) for unidirectional ratcheted key exchange (URKE) schemes, in particular by defining an appropriate set of trivial attacks. We propose a construction that we prove secure under this notion based on a leakage-resilient one-time MAC and a KUOWR-secure kuKEM. By instantiating this MAC in the random oracle model, results from Balli, Rösler and Vaudenay [BRV20a] imply that LR-KIND URKE and KUOWR-secure kuKEM (and therefore regular KIND URKE) are equally powerful.
- To overcome the inherent limitations of LR-KIND security, Section 5 introduces a security notion capturing one-wayness for URKE under leakage (LR-OW security), together with a new set of trivial attacks. We provide a construction secure under this notion that uses leakage-resilient kuKEM.
- Towards building leakage-resilient kuKEM, in Section 6.1, we reduce the problem of constructing leakage-resilient kuKEM to building one-way, CPA-secure leakage-resilient hierarchical identity-based encryption.
- In Section 6.2, we show that, given a random oracle and leakage-resilient one-time MAC, leakage-resilient kuKEM can be built from one-way leakage-resilient URKE (and vice-versa from our results from Sections 4 and 5). That is, we show that LR-OW-secure URKE and LR-KUOWR-secure kuKEM are equally powerful in the random oracle model.

1.2 Technical overview

URKE security. To define security for leakage-resilient unidirectional ratcheted key exchange (URKE), we build on the notion of key indistinguishability under randomness manipulation introduced by Balli, Rösler and Vaudenay [BRV20a]. Our security notions therefore allow adversarial manipulation of randomness, which makes them more general in this aspect [ACD19; BRV20a] compared to those that do not [BOS17; PR18]. Given that we model partial leakage, incorporating randomness leakage also appears more realistic.

For the sake of readability and modularity, we use a different formalism than Balli, Rösler and Vaudenay [BRV20a], that enables us to reduce the number of book-keeping variables and checks in oracles by pushing most of the logic to a *trivial* function which checks at the end of a game run if any of the constraints enforced within Balli, Rösler and Vaudenay's oracles were violated, and aborts the execution if any were.

In more detail, we opted for a very general notion of leakage, which (ignoring restrictions due to trivial attacks) enables the adversary to leak a certain number of bits of its choice on the secret states after each Send or successful Receive operation. There is therefore a bound on leakage per such operation, but no global bound on leakage during the whole game. This is a variant of continual leakage as described in [KR19; Dod+10b]. This is reasonable, since most side-channels only occur if there are computations involving the secret states. In cases where continual leakage is impossible, we also consider a second and simpler notion, called bounded leakage, where the bound is never reset and the amount of leakage in the whole game is therefore bounded by it.

In both cases, only secret keys and states are leaked, placing our leakage notion withing the category of leakage on memory as defined in [KR19]. This contrasts with leakage on computation, where intermediary values from potentially randomized computations are also exposed. However, the possibility of randomness manipulation allows the adversary to gain information about computation results as well.

In order to be as general as possible, we allow the adversary to adaptively obtain one bit after the other up to the per-computation leakage bound. To obtain one bit of information, it chooses an efficiently computable leakage function f which on any input returns a value in $\{0, 1\}$. It then receives the output of f on the chosen secret state or key. We assume that leaking cannot modify the secret states or keys. With this approach to leakage we follow [Dzi06].

Key indistinguishability. Our first notion security notion for URKE, a key indistinguishability notion (LR-KIND), requires the adversary to distinguish between uniformly random keys and keys output by the protocol execution, generalising the original game from Balli, Rösler and Vaudenay

However, with our definition of leakage, leaking even just one bit on the receiving party B allows an adversary to learn a bit of the challenge key, by simulating the reception of a challenge key ciphertext within the leakage function. The challenge key message can be obtained by the adversary before it is received as we assume that the adversary has full network control. This is possible even if the Receive algorithm or the state updates of B are randomized, since it exploits the correctness of the scheme. Therefore, a key indistinguishability notion for URKE cannot permit any leakage on the receiving party B beyond leakage that is allowed when state exposure is also allowed. Leakage on A, on the other hand, is allowed except when 1) A's state is leaked beyond the leakage bound; 2) the leakage is then used to impersonate towards B; *and then* 3) the resulting key derived by B is challenged.

In Section 4, after defining our LR-KIND notion, we present a URKE scheme secure under this notion from a leakage-resilient one-time strongly-unforgeable (LR-OT-SUF) MAC scheme and a kuKEM scheme that provides one-wayness under randomness manipulation (KUOWR) [BRV20a; RSS23]. In particular, since the LR-KIND security notion does not allow leakage on the receiver B anyway, there is no need for the kuKEM to be leakage-resilient. This is prudent for performance as kuKEM is a more powerful and expensive primitive. Given that Balli, Rösler and Vaudenay showed that secure *kuKEM* can be used to build KIND-secure URKE and vice-versa given a one-time MAC and random oracle [BRV20a], our construction implies the same for secure kuKEM and LR-KIND URKE in the random oracle model (given a random oracle instantiation of the MAC). Note that this does not imply that *every* KIND-secure URKE is also LR-KIND: we argue that Balli, Rösler and Vaudenay's construction is not in general LR-KIND-secure in Appendix A.

One-wayness. However, as explained above, in order to define our achievable LR-KIND notion, we needed to restrict the adversary in many ways. In particular, leakage on the receiver and on the challenge key are not allowed. Therefore, this notion might not capture the abilities of side-channel attacks or threats in general that leak on the receiver side.

To overcome this, we formalise a one-wayness (OW) security notion for URKE that allows us to relax the restrictions from our LR-KIND game. One-wayness captures the guarantee that an efficient adversary can obtain a complete challenge key only with negligible probability. It is not an indistinguishability notion, and therefore in some aspect weaker. However, it enables us to allow for bounded (but not continual) leakage on the receiver. This means that between the beginning of the game and the reception of the challenge key, only a fixed number of bits can be leaked on the state of B. This restriction is again due to the "leak-into-the-future" attack which was already a problem for the LR-KIND definition. We therefore define the LR-OW security notion and provide a construction achieving it in the random oracle model, which requires a bounded leakage-resilient (LR-KUOWR)-secure kuKEM and a leakage-resilient MAC. In particular, leakage-resilient kuKEM generalises kuKEM to additionally allow bounded secret key leakage.

On the complexity of leakage-resilient kuKEM. Given our new one-wayness notion relies on a new primitive, namely leakage-resilient kuKEM, we then consider how costly constructing it is. It is known how to construct (regular, KUOWR-secure) kuKEM either from bounded-collusion identity-based encryption (IBE) (supporting bounded usage), or from IND-CCA-secure [RSS23] or OW-CCA-secure hierarchical IBE (HIBE) [BRV20a]. In Section 6.1, we construct a LR-KUOWR-secure kuKEM assuming a LR-OW-CCA-secure HIBE. In particular, we show that an LR-OW-CCA-secure HIBE scheme can be constructed from LR-OW-CPA-secure HIBE and true-simulation extractable NIZKs [Dod+10c]. In the LR-OW-CPA game, the adversary has to decrypt a random message encrypted under a chosen identity vector even when they can leak some bounded amount of information on secret keys, and can expose secret keys that cannot be used to trivially decrypt the challenge. In LR-OW-CCA, the adversary is additionally given access to a decryption oracle. We then discuss and leave open the problem of constructing LR-KUOWR-secure kuKEM directly.

Recall that Balli, Rösler and Vaudenay [BRV20a] showed that KUOWR-secure kuKEM, relative to a random oracle and MAC, is as powerful as a KIND-secure URKE. We port these results to the leakage-resilient setting, namely between LR-OW-secure URKE and LR-KUOWR-secure kuKEM given a random oracle and a leakage-resilient (LR-OT-SUF-secure) one-time MAC. To this end, in Section 6.2 we construct LR-KUOWR-secure kuKEM *from* a random oracle, a leakage-resilient MAC and LR-OW-secure URKE. By comparison to Balli, Rösler and Vaudenay, we use an additional random oracle to hash the MAC key output by the URKE. Given our construction of LR-OW-secure URKE from LR-KUOWR-secure kuKEM, this implies that LR-OW-secure URKE and LR-KUOWR-secure kuKEM are equally powerful in the random oracle model.

Looking forward. Our work opens the pathway for providing strong guarantees under partial leakage in messaging in theory and practice. Future directions of research include extending our work to capture bidirectional ratcheting where both parties can send and receive, messaging proper, group ratcheting or messaging, and exploring alternate notions of leakage and performance/security trade-offs.

1.3 Additional related work

Providing security guarantees for messaging in the presence of state exposure (i.e., forward and post-compromise security) has been explored in theory and practice alike in both the two-party and the more complex group settings. Many works provide different trade-offs between performance and security: in the two-party literature, for example, some protocols require relatively heavy primitives like kuKEM (e.g., [JS18; PR18; BRV20a] and ours), and conversely some protocols only require symmetric cryptography [YV20; Yan+23]. Other problems related to messaging like active attack detection [Bar+23], handling multiple devices securely [Dim+21] and the modelling of practical schemes [ACD19; Alw+20a; BCG23; ADJ24] have been considered (to name but a few).

In the literature on leakage resilience, several different types and measures of leakage have been proposed, including entropy-bounded leakage (rather than space-bound), continual leakage and after-the-fact leakage [HL11] (modelling leakage after challenging), and many cryptographic primitives have been augmented to provide leakage resilience. We refer the reader to [KR19] for a (non-exhaustive) summary of the literature. As our model allows arbitrary but bounded leakage on the entire secret state, it is stronger than works that assume that “only computation leaks” (OCL) [MR04], i.e., only the part of the secret state used in a given computation leaks, which may nonetheless be of interest for performance reasons.

Previous work has considered leakage resilience in the context of authenticated key exchange (AKE) protocols, by contrast to ratcheted key exchange which assumes at initialisation that some initial key exchange has already taken place. Works on AKE have explored protecting long-term secrets against leakage and full ephemeral state exposure [ADW09a; CPR17], as well as against leakage on both long-term and ephemeral secrets [Yan+19]. The work of [Bel+17; Cha+23] consider forward-secure primitives like signatures and public-key encryption under continual leakage, and so they also provide guarantees under both full and partial state exposure but only for one party, the secret key holder. We note that leakage-resilient authenticated encryption with associated data (AEAD) in particular has been considered, from which leakage-resilient messaging could be constructed using leakage-resilient ratcheted key exchange.

In recent years, we have seen an increase in software-visible side channels, such as Spectre [Koc+19], Meltdown [Lip+18] and Hertzbleed [Wan+22], just to name a few. These and similar attacks [Gen+16] represent a serious threat to actual system, both personal computers and mobile phones, as microprocessors from AMD, Intel and ARM are found to be vulnerable. Some of these attacks allow for repeated leakage on information on secret data, but not necessarily all of it at once. They therefore justify the need for cryptographic primitives which can withstant the repeated exposure of a small subset of their secrets.

2 Preliminaries

Let λ be the security parameter (sometimes written in unary as 1^λ). We use a general key space $\mathcal{K}$ for all primitives for simplicity. An efficient algorithm is one that is probabilistic polynomial-time. We consider two parties, A and B; if $\mathcal{P}$ is one party (A resp. B), $\overline{\mathcal{P}}$ is their partner (B resp. A). For strings, we denote concatenation with $||$, so that we have $\epsilon \mid\mid X = X$ for empty string ϵ and any string X, the prefix-of relation with $\prec$, the equal-or-prefix-of relation with $\preceq$ and the not-prefix-of notation with $\not\preceq$ (see Appendix A for a formal definition of the relations). Inclusion of one string in another is denoted by $\in$: $a \in b$ means that the string b contains all characters of a in the same order. An algorithm can abort: for indistinguishability games, we denote this by $\textbf{abort}_{\mathsf{IND}}$ and otherwise by **abort**. These abortions are defined so that they cannot be exploited by an adversary for increasing its winning probability.

All oracles that enable the adversary to apply a leakage function to some party's secret take as input a leakage function f, whose codomain must be the binary set. For clarity, our games do not check this constraint explicitly. To enforce an explicit check, one could add a special oracle to all security games, which checks whether leakage functions respect the constraint. Reductions in proofs would then outsource this check to the special oracle in order to stay efficient themselves. It is important that the oracle does not return a value which is not a bit in secret-dependent cases, as this would allow to learn more than one bit of information per call to the leak oracle. Even though the oracle executes f, the function is chosen by the adversary, and we therefore count its runtime and the random oracle calls it makes towards the runtime and random oracle calls of the adversary. The leakage functions can query only random oracles and not any other oracles in our games.

We provide more detailed preliminaries including precise notations and abortion definitions in Appendix A.

3 Cryptographic primitives

This section introduces the cryptographic primitives on which this work relies. We start with message authentication code (MAC) schemes and key-updatable KEM (kuKEM) schemes [BRV20a]; for both schemes we formalize security in the bounded-leakage model, which assumes a bounded amount of leakage. We then turn our attention to unidirectional ratcheted key exchange (URKE).

In this paper we use leakage-resilient MAC, and classic and leakage-resilient kuKEM schemes to construct URKE schemes which achieve different notions of security. To ease the composition of these schemes, we work with a general key space for each security parameter λ, by assuming a deterministic function $\mathsf{keyspace}$ which inputs a security parameter λ and outputs a key space $\mathcal{K}$.

3.1 Message authentication code

In this section we introduce message authentication code (MAC) schemes.

Definition 1 (MAC). *A* message authentication code *(MAC) comprises the following efficient algorithms:*

- $\mathsf{Gen}(1^\lambda) \to \mathsf{k}$ *takes a unary string* 1^λ *and outputs a key* k*.*
- $\mathsf{Tag}(\mathsf{k}, m) \to \mathsf{t}$ *takes a key* k *and a message* m *and returns a tag* t*.*
- $\mathsf{Ver}(\mathsf{k}, m, \mathsf{t}) \to \mathsf{acc}$ *takes a key* k*, a message* m *and a tag* t *and returns an acceptance bit* $\mathsf{acc} \in \{\mathsf{true}, \mathsf{false}\}$*.*

We assume that the generation of a MAC key $\mathsf{Gen}(1^\lambda)$ *consists in the uniformly random choice of* $\mathsf{k} \leftarrow\$ \mathcal{K}$*, where* $\mathcal{K} \leftarrow \mathsf{keyspace}(\lambda)$ *is our general key space.*

Informally, a MAC scheme is *correct* if the verification of a message m and a tag generated on the same message m always succeed. We formally define this property in Definition 2.

Definition 2 (MAC correctness). *A MAC scheme* $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *is* correct *if for all security parameters* $\lambda \in \mathbb{N}$ *the following holds:*

$$\Pr[\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda); \mathsf{Ver}(\mathsf{k}, m, \mathsf{Tag}(\mathsf{k}, m)) \to \mathsf{true}] = 1,$$

where the probability is taken over all the random coins used by the algorithms of the scheme.

This has the advantage that the adversary cannot outsource costly computations to the leak oracles in order to improve its own runtime.

We formalize security for MAC with the notion of one-time strong unforgeability, formalized through the LR-OT-SUF game in Fig. 1. In the game, the adversary can generate—through the TAG oracle—and verify—through the VER oracle—a tag for an arbitrary message. We formalize the leakage-resilient property of this security notion with the LEAK-KEY oracle, which enables the adversary to leak a function f of the key k, provided that the function returns a single bit. The adversary wins by forging a valid message and tag pair if the message/tag pair was not generated by a query to TAG.

```
Game $\mathsf{LR\text{-}OT\text{-}SUF}^{\ell}_{\mathcal{A}}(1^\lambda)$
1 :  $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$
2 :  $Q \leftarrow \perp;\ q \leftarrow 0$
3 :  $(\mathsf{pt}^*, \mathsf{t}^*) \leftarrow \mathcal{A}^{\mathsf{TAG},\mathsf{VER},\mathsf{LEAK\text{-}KEY}}(1^\lambda)$
4 :  if $(\mathsf{pt}^*, \mathsf{t}^*) = Q$ then return $0$
5 :  if $\neg\mathsf{Ver}(\mathsf{k}, \mathsf{pt}^*, \mathsf{t}^*)$ then return $0$
6 :  return $1$

Oracle $\mathsf{VER}(\mathsf{pt}, \mathsf{t})$
1 :  return $\mathsf{Ver}(\mathsf{k}, \mathsf{pt}, \mathsf{t})$

Oracle $\mathsf{TAG}(\mathsf{pt})$
1 :  if $Q \neq \perp$ then abort
2 :  $\mathsf{t} \leftarrow \mathsf{Tag}(\mathsf{k}, \mathsf{pt})$
3 :  $Q \leftarrow (\mathsf{pt}, \mathsf{t})$
4 :  return $\mathsf{t}$

Oracle $\mathsf{LEAK\text{-}KEY}(f)$
1 :  if $q > \ell$ then abort
2 :  $q \leftarrow q + 1$
3 :  return $f(\mathsf{k})$
```

Fig. 1: LR-OT-SUF game. Classic OT-SUF security for MAC schemes is defined using the same game except that the adversary cannot query the LEAK-KEY oracle (equivalent to $\ell = 0$).

Definition 3 (LR-OT-SUF). *We say that a MAC scheme* MAC $=$ (Gen, Tag, Ver) *is* (q, ℓ, t, ϵ)-LR-OT-SUF *secure for security parameter* λ *if, for all adversaries* $\mathcal{A}$ *which make at most* q *oracle queries and run in time at most* t*, we have:*

$$\Pr[\mathsf{LR\text{-}OT\text{-}SUF}^{\ell}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] \leq \epsilon,$$

where the probability is taken over all the random coins that the challenger and the adversary use and the game LR-OT-SUF *is defined in Fig. 1.*

Assuming keyspace $\mathcal{K}$ and a random oracle H with inputs in $\mathcal{K} \times \{0,1\}^*$ and outputs in $\mathcal{K}$ we can define a MAC scheme as follows: Gen outputs a uniformly random output k_m in $\mathcal{K}$. $\mathsf{Tag} : \mathcal{K} \times \{0,1\}^* \to \mathcal{K}$ is identical to querying H on its inputs. $\mathsf{Ver} : \mathcal{K} \times \{0,1\}^* \times \mathcal{K} \to \{0,1\}$ queries H on its first 2 inputs and tests whether the output is equal to its third input. Since the random oracle queries to H are counted in the total number of queries q, this MAC scheme is $(q, \ell, t, q \cdot 2^\ell / |\mathcal{K}|)$-LR-OT-SUF secure. This is because given the knowledge of the chosen message and the ℓ leaked bits of the key in $|\mathcal{K}|$, guessing the complete input of H correctly has probability $2^\ell/|\mathcal{K}|$, and the adversary makes at most

$q-1$ attemps of getting a tag via such guessing (since it makes at most $q-1$ queries to H). The probability of directly guessing the tag is $1/|\mathcal{K}|$.

In the standard model, an example of an $\mathsf{LR\text{-}OT\text{-}SUF}$-secure MAC scheme can be found in the work of Hazay, López-Alt, Wee and Wichs [Haz+16, Section 5.3].

3.2 Key-updatable key encapsulation mechanism (kuKEM)

Key-updatable key encapsulation mechanisms (kuKEM) were first introduced by Poettering and Rösler [PR18] and later adapted by Balli, Rösler and Vaudenay [BRV20a].

Similar to a standard KEM scheme, a kuKEM scheme establishes secure symmetric keys between a public key and secret key holder. The encapsulation algorithm takes a public key and outputs an updated public key, an encapsulated key and a ciphertext. The decapsulation mechanism inputs a secret key and a ciphertext and outputs an updated secret key and either a decapsulated key or an error symbol. A kuKEM scheme possesses an additional feature: a pair of algorithms (UpPk and UpSk) that generate new public/secret keys, to which we refer to as "updated" keys, based on existing ones and associated data. Importantly, a key pair retains its functionality given that the key pair is updated consistently, i.e., the update functions are called with the same associated data, and encapsulation and decapsulation are called consistently.

Definition 4 (kuKEM [BRV20a]). *A* key-updatable key encapsulation mechanism *(kuKEM) comprises the following efficient algorithms:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ *takes a unary string* 1^λ *and outputs public parameters* pp*.*
- $\mathsf{Gen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$ *takes public parameters* pp *and outputs initial public and secret key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Encaps}(\mathsf{pk}) \to (\mathsf{pk}', \mathsf{k}, \mathsf{ct})$ *takes a public key* pk *and outputs a new public key* pk'*, encapsulated key* k *and ciphertext* ct*.*
- $\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}) \to (\mathsf{sk}', \mathsf{k})$ *takes a secret key* sk *and a ciphertext* ct *and outputs a new secret key* sk' *and either a decapsulated key* k *or a bottom value* $\perp$ *that denotes failure.*
- $\mathsf{UpPk}(\mathsf{pk}, \mathsf{ad}) \to \mathsf{pk}'$ *takes a public key* pk *and associated data* ad *and outputs a new public key* pk'*.*
- $\mathsf{UpSk}(\mathsf{sk}, \mathsf{ad}) \to \mathsf{sk}'$ *takes a secret key* sk *and associated data* ad *and outputs a new secret key* sk'*.*

Exchanged keys k *and associated data* ad *are elements of the general key space* $\mathcal{K} \leftarrow \mathsf{keyspace}(\lambda)$*.*

We refer the reader to the work of Balli, Rösler and Vaudenay for a definition of correctness [BRV20a, Definition 1]. Informally, we require that if pk and sk are updated consistently (through Encaps and UpPk for pk and Decaps and UpSk for sk), then Encaps and Decaps output the same key k when queried with consistent input.

```
Game $\mathsf{KUOWR}_{\mathcal{A}}(1^\lambda)$  $\mathsf{LR\text{-}KUOWR}^{\ell}_{\mathcal{A}}(1^n)$
    $\mathsf{win} \leftarrow 0$
    $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{pp})$
    $\mathsf{CK}[\cdot] \leftarrow \perp;\ XP, \mathsf{tr_A}, \mathsf{tr_B} \leftarrow \perp$
    $c \leftarrow 0$
    $\mathsf{SK}[\cdot] \leftarrow \perp;\ \mathsf{SK}[\mathsf{tr_B}] \leftarrow \mathsf{sk}$
    $\mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{pk})$
    return $\mathsf{win}$

$\mathsf{EXP}(\mathsf{tr})$
    if $\mathsf{SK}[\mathsf{tr}] = \perp$ then return
    $XP \leftarrow XP \cup \{\mathsf{tr}^* : \mathsf{tr} \prec \mathsf{tr}^*\}$
    return $\mathsf{SK}[\mathsf{tr}]$

$\mathsf{SOLVE}(\mathsf{tr}, \mathsf{k})$
    if $\mathsf{tr} \in XP$ then return $\perp$
    if $\mathsf{CK}[\mathsf{tr}] = \perp$ then return $\perp$
    if $\mathsf{k} = \mathsf{CK}[\mathsf{tr}]$ then $\mathsf{win} \leftarrow 1$
    return $\mathsf{win}$

$\mathsf{LEAKSK}(\mathsf{tr}, f)$
    if $c \geq \ell$ then return $\perp$
    $c \leftarrow c + 1$
    return $f(\mathsf{SK}[\mathsf{tr}])$

$\mathsf{ENC}(\mathsf{r})$
    $\mathsf{fresh} \leftarrow \mathsf{false}$
    if $\mathsf{r} = \perp$ then $\mathsf{fresh} \leftarrow \mathsf{true}$
    if $\mathsf{fresh}$ then $\mathsf{r} \leftarrow\$ \mathcal{R}$
    $(\mathsf{pk}, \mathsf{k}, \mathsf{ct}) \leftarrow \mathsf{Encaps}(\mathsf{pk}; \mathsf{r})$
    $\mathsf{tr_A} \leftarrow \mathsf{tr_A} \mid\mid (ct : \mathsf{ct})$
    if $\mathsf{fresh}$ then $\mathsf{CK}[\mathsf{tr_A}] \leftarrow \mathsf{k}$
    return $(\mathsf{pk}, \mathsf{ct})$

$\mathsf{DEC}(\mathsf{ct})$
    $(\mathsf{sk}, \mathsf{k}) \leftarrow \mathsf{Decaps}(\mathsf{sk}, \mathsf{ct})$
    if $\mathsf{k} = \perp$ then return $\perp$
    $\mathsf{tr_B} \leftarrow \mathsf{tr_B} \mid\mid (ct : \mathsf{ct})$
    $\mathsf{SK}[\mathsf{tr_B}] \leftarrow \mathsf{sk}$
    if $\mathsf{CK}[\mathsf{tr_B}] \neq \perp$ then return $\perp$
    return $k$

$\mathsf{UPPK}(\mathsf{ad})$
    $\mathsf{pk} \leftarrow \mathsf{UpPk}(\mathsf{pk}, \mathsf{ad})$
    $\mathsf{tr_A} \leftarrow \mathsf{tr_A} \mid\mid (ad : \mathsf{ad})$
    return $\mathsf{pk}$

$\mathsf{UPSK}(\mathsf{ad})$
    $\mathsf{sk} \leftarrow \mathsf{UpSk}(\mathsf{sk}, \mathsf{ad})$
    $\mathsf{tr_B} \leftarrow \mathsf{tr_B} \mid\mid (ad : \mathsf{ad})$
    return
```

Fig. 2: Single-user KUOWR kuKEM security notion adapted from [BRV20a]. The array CK stores challenge keys, XP holds exposed secret keys, and $\mathsf{tr_A}$, $\mathsf{tr_B}$ track transcripts (see [BRV20a] further details on these variables). The oracle set is $\mathcal{O} = (\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPPK}, \mathsf{UPSK}, \mathsf{EXP}, \mathsf{SOLVE})$, and for $\mathsf{LR\text{-}KUOWR}$ the same together with LEAKSK.

We formalize security for kuKEM with the notion of one-way security under randomness manipulation in a single-instance setting, or KUOWR. We also formalize leakage-resilient KUOWR security which we denote by $\mathsf{LR\text{-}KUOWR}$. The only difference between the KUOWR and $\mathsf{LR\text{-}KUOWR}$ games is that while playing the latter the adversary has access to an additional leaking oracle LEAKSK. We define single-instance rather than multi-instance security as done by Balli, Rösler and Vaudenay [BRV20a] since all the security notions that we define in this work are in the single-instance setting. A standard guessing technique can be used to reduce security with a factor of q security loss given that the key generation oracle is called q times in the multi-instance game.

Definition 5 ($\mathsf{KUOWR}/\mathsf{LR\text{-}KUOWR}$). *We say that a kuKEM Π is (q, t, ϵ)-*KUOWR *(resp. (q, ℓ, t, ϵ)-*$\mathsf{LR\text{-}KUOWR}$*) secure for security parameter λ if, for all adversaries $\mathcal{A}$ which make at most q oracle queries and run in time at most t, we have:*

$$\Pr[\mathsf{KUOWR}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] \leq \epsilon \ (\textit{resp.}\ \Pr[\mathsf{LR\text{-}KUOWR}^{\ell}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] \leq \epsilon),$$

where the probability is taken over all the random coins that the challenger and the adversary use and the game KUOWR *(resp.* $\mathsf{LR\text{-}KUOWR}$*) is defined in Fig. 2 for $\mathcal{O} = (\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPPK}, \mathsf{UPSK}, \mathsf{EXP}, \mathsf{SOLVE})$ (resp. $\mathcal{O} = (\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPPK}, \mathsf{UPSK}, \mathsf{EXP}, \mathsf{SOLVE}, \mathsf{LEAKSK})$).*

Constructing leakage-resilient kuKEM. In Section 6.1, we construct $\mathsf{LR\text{-}KUOWR}$-secure kuKEM from hierarchical identity-based encryption and and non-interactive zero-knowledge proofs (NIZKs) and discuss additional constructions.

3.3 Unidirectional ratcheted key exchange

We now introduce unidirectional ratcheted key exchange (URKE). In this primitive, first introduced by Bellare, Camper Singh, Jaeger, Nyayapati, and Stepanovs [Bel+17], the roles of the two parties are disjoint: A is the sender and B the receiver.

Definition 6 (Unidirectional ratcheted key exchange (URKE)). *A* unidirectional ratcheted key exchange *(URKE) comprises the following efficient algorithms:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ *takes the security parameter $\lambda \in \mathbb{N}$, expressed in unary, and outputs public parameters* pp.
- $\mathsf{Init}(\mathsf{pp}) \to (\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B}, z)$ *takes public parameters* pp *and outputs a state* $\mathsf{st}_\mathcal{P}$ *for $\mathcal{P} \in \{\mathsf{A}, \mathsf{B}\}$ and public information z.*
- $\mathsf{Send}(\mathsf{st}_\mathsf{A}, \mathsf{ad}) \to (\mathsf{st}'_\mathcal{P}, \mathsf{ct}, \mathsf{k})$ *takes a state* st_A*, associated data* ad *and a plaintext* pt *and outputs a new state* st'_A*, ciphertext* ct *and key* k.
- $\mathsf{Receive}(\mathsf{st}_\mathsf{B}, \mathsf{ad}, \mathsf{ct}) \to (\mathsf{acc}, \mathsf{st}'_\mathsf{B}, \mathsf{k})$ *takes a state* st_B*, associated data* ad *and ciphertext* ct *and outputs an acceptance bit* $\mathsf{acc} \in \{\mathsf{true}, \mathsf{false}\}$*, state* st'_B *and key* k.

When $\mathsf{acc} = \mathsf{false}$*, the* $\mathsf{Receive}$ *algorithm returns* $\mathsf{st}'_\mathsf{B} \leftarrow \mathsf{st}_\mathsf{B}$ *and* $\mathsf{k} \leftarrow \perp$.

Intuitively, a URKE scheme is *correct* if the Send and $\mathsf{Receive}$ algorithms return the same key k on consistent inputs.

More precisely, correctness for URKE schemes builds on two oracles that we introduce in Fig. 3. The SEND oracle enables the adversary to send a message on behalf of party A, either by specifying the randomness that the Send algorithm uses or by leaving the oracle to sample randomness uniformly. The $\mathsf{RECEIVE}$

```
Oracle $\mathsf{SEND}(\mathsf{ad}, \mathsf{r})$
1 : $i \leftarrow i + 1$
2 : $\mathsf{r}_{\mathsf{in}} \leftarrow \mathsf{r}$
3 : if $\mathsf{r} = \perp$ then $\mathsf{r} \leftarrow\!\!\$\, \mathcal{R}$
4 : $(\mathsf{st}_\mathsf{A}, \mathsf{ct}, \mathsf{k}) \leftarrow \mathsf{Send}(\mathsf{st}_\mathsf{A}, \mathsf{ad}; \mathsf{r})$
5 : $\mathsf{tr}_\mathsf{A} \leftarrow \mathsf{tr}_\mathsf{A} \mid\mid (\mathsf{ad}, \mathsf{ct})$
6 : $\mathsf{state}[\mathsf{A}, \mathsf{tr}_\mathsf{A}] \leftarrow \mathsf{st}_\mathsf{A}$
7 : $\mathsf{key}[\mathsf{A}, \mathsf{tr}_\mathsf{A}] \leftarrow \mathsf{k}$
8 : $\mathsf{log}[i] \leftarrow (\text{“send”}, \mathsf{tr}_\mathsf{A}, \mathsf{ad}, \mathsf{ct}, \mathsf{r}_{\mathsf{in}}, \mathsf{k})$
9 : return $\mathsf{ct}$
```

```
Oracle $\mathsf{RECEIVE}(\mathsf{ad}, \mathsf{ct})$
1 : $i \leftarrow i + 1$
2 : $(\mathsf{acc}, \mathsf{st}, \mathsf{k}) \leftarrow \mathsf{Receive}(\mathsf{st}_\mathsf{B}, \mathsf{ad}, \mathsf{ct})$
3 : if $\neg\mathsf{acc}$ then
4 :     $\mathsf{log}[i] \leftarrow (\text{“failedrec”}, \mathsf{tr}_\mathsf{B}, \mathsf{ad}, \mathsf{ct})$
5 :     return $\perp$
6 : $\mathsf{st}_\mathsf{B} \leftarrow \mathsf{st}$
7 : $\mathsf{tr}_\mathsf{B} \leftarrow \mathsf{tr}_\mathsf{B} \mid\mid (\mathsf{ad}, \mathsf{ct})$
8 : $\mathsf{state}[\mathsf{B}, \mathsf{tr}_\mathsf{B}] \leftarrow \mathsf{st}_\mathsf{B}$
9 : $\mathsf{key}[\mathsf{B}, \mathsf{tr}_\mathsf{B}] \leftarrow \mathsf{k}$
10 : $\mathsf{log}[i] \leftarrow (\text{“rec”}, \mathsf{tr}_\mathsf{B}, \mathsf{ad}, \mathsf{ct}, \mathsf{k})$
11 : return
```

Fig. 3: URKE oracles which use variables state, tr_*, key, log, st_* and i that are all initialized in games where the oracles are used.

oracle enables the adversary to receive a message for party B. Both oracles use the *transcript* or *trace* of a party $\mathcal{P} \in \{\mathsf{A}, \mathsf{B}\}$ to index maps that store states and keys of $\mathcal{P}$, as well as the map log to record oracles calls.

We formally capture this in the $\mathsf{CORRECT}$ game of Fig. 4. In the game, the adversary has access to the oracles SEND and $\mathsf{RECEIVE}$; the predicate $\mathsf{different\text{-}key}$ enforces that the scheme's algorithms produce identical keys on consistent transcripts, whereas $\mathsf{incorrect\text{-}reject}$ ensures that the $\mathsf{Receive}$ algorithm correctly accepts every $(\mathsf{ad}, \mathsf{ct})$ pair that the Send algorithm returns.

Definition 7 (URKE correctness). *Consider the* $\mathsf{CORRECT}$ *game of Fig. 4. A URKE scheme is* correct *if for all (possibly unbounded) adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr[\mathsf{CORRECT}^{\mathcal{A}}(1^\lambda) \Rightarrow 1] = 0.$$

4 Leakage-resilient key indistinguishability for URKE

In this section we introduce leakage-resilient key indistinguishability for URKE, which we denote $\mathsf{LR\text{-}KIND}$. We first introduce the security notion and then present a URKE construction that achieves $\mathsf{LR\text{-}KIND}$ security assuming a classic kuKEM scheme and a leakage-resilient MAC scheme.

4.1 Security definition

We introduce our notion of leakage-resilient key indistinguishability for URKE schemes. The formal definition relies on a set of oracles, introduced in Fig. 5. The oracles SEND and $\mathsf{RECEIVE}$ are the same as in Fig. 3: we report them here

```
Game $\mathsf{CORRECT}^{\mathcal{A}}(1^\lambda)$
1 : $(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda);\ (\mathsf{st_A}, \mathsf{st_B}, z) \leftarrow \mathsf{Init}(\mathsf{pp})$
2 : $\mathsf{st}_*, \mathsf{tr}_* \leftarrow \bot;\ \mathsf{state}[\cdot], \mathsf{key}[\cdot], \mathsf{log}[\cdot] \leftarrow \bot$
3 : $i \leftarrow 0$
4 : $\mathcal{A}(\mathsf{pp}, z)^{\mathsf{SEND},\mathsf{RECEIVE}}$
5 : if $\mathsf{different\text{-}key}(\mathsf{log}) \vee \mathsf{incorrect\text{-}reject}(\mathsf{log})$ then
6 :     return 1
7 : return 0

$\mathsf{different\text{-}key}(\mathsf{log})$
1 : return $\exists\, \mathsf{tr}, i, j, \mathsf{k_A}, \mathsf{k_B}, \mathsf{ad}, \mathsf{ct}$ :
2 :     $\mathsf{log}[i] = (\text{“send”}, \mathsf{tr}, \mathsf{ad}, \mathsf{ct}, \cdot, \mathsf{k_A}) \wedge \mathsf{log}[j] = (\text{“rec”}, \mathsf{tr}, \mathsf{ad}, \mathsf{ct}, \mathsf{k_B}) \wedge (\mathsf{k}_A \neq \mathsf{k_B})$

$\mathsf{incorrect\text{-}reject}(\mathsf{log})$
1 : return $\exists\, \mathsf{tr}, i, j, \mathsf{ad}, \mathsf{ct}$:
2 :     $\mathsf{log}[i] = (\text{“send”}, \mathsf{tr}, \mathsf{ad}, \mathsf{ct}, \cdot, \cdot) \wedge \mathsf{log}[j] = (\text{“failedrec”}, \mathsf{tr}, \mathsf{ad}, \mathsf{ct})$
```

Fig. 4: Correctness game for a URKE scheme.

for the reader's convenience. The oracles EXP-STATE and EXP-KEY expose the state and key of party $\mathcal{P}$, respectively. The two leakage oracles, LEAK-STATE and LEAK-KEY, take as inputs a leakage function f, a party $\mathcal{P}$ and a trace tr; both abort if the function's codomain is not the binary set $\{0, 1\}$. The oracle LEAK-STATE enables the adversary to apply the leakage function to the state of a party, whereas LEAK-KEY evaluates the function on the key of party $\mathcal{P}$. Even tough the leakage function's output is a single bit, the adversary can leak on functions with multiple bits of output (subject to the leakage bound) by querying the leakage oracles multiple times.

In Fig. 6 we introduce the game LR-KIND, which formalizes the notion of leakage-resilient key indistinguishability that we define in Definition 8. In this game the adversary has access to a set of oracles $\mathcal{O}$ and must distinguish between a key that Send or Receive returns and a key randomly generated by the challenger. Without any restriction, the set of oracles gives the adversary the chance of trivially winning, i.e., trivially distinguish a challenge key from a random key. These attacks exploit direct exposures of secrets—e.g., exposing and challenging the same secret key—and the correctness guarantees of the scheme—e.g., exposing the key that A generates and challenging the same key that B outputs. Similarly to previous works on ratcheted key exchange [Bel+17; PR18; BRV20a], we rule out these, and only these, attacks. We refer to this minimal restriction on the adversarial behaviour as *optimal*, because we restrict the adversary to what is necessary for a meaningful security definition and nothing more. Concretely, we exclude the trivial attacks with the trivial-KIND predicate in line 8 in LR-KIND

```
Oracle SEND(ad, r)
  1: $i \leftarrow i + 1$
  2: $\mathsf{r}_{\mathsf{in}} \leftarrow \mathsf{r}$
  3: if $\mathsf{r}_{\mathsf{in}} = \bot$ then $\mathsf{r}_{\mathsf{in}} \leftarrow\!\!\$\ \mathcal{R}$
  4: $(\mathsf{st}_\mathsf{A}, \mathsf{ct}, \mathsf{k}) \leftarrow \mathsf{Send}(\mathsf{st}_\mathsf{A}, \mathsf{ad}; \mathsf{r}_{\mathsf{in}})$
  5: $\mathsf{tr}_\mathsf{A} \leftarrow \mathsf{tr}_\mathsf{A} \,||\, (\mathsf{ad}, \mathsf{ct})$
  6: $\mathsf{state}[\mathsf{A}, \mathsf{tr}_\mathsf{A}] \leftarrow \mathsf{st}_\mathsf{A}$
  7: $\mathsf{key}[\mathsf{A}, \mathsf{tr}_\mathsf{A}] \leftarrow \mathsf{k}$
  8: $\mathsf{log}[i] \leftarrow (\text{“send”}, \mathsf{tr}_\mathsf{A}, \mathsf{ad}, \mathsf{ct}, \mathsf{r})$
  9: return ct

Oracle RECEIVE(ad, ct)
  1: $(\mathsf{acc}, \mathsf{st}, \mathsf{k}) \leftarrow \mathsf{Receive}(\mathsf{st}_\mathsf{B}, \mathsf{ad}, \mathsf{ct})$
  2: if $\neg\mathsf{acc}$ then return $\bot$
  3: $i \leftarrow i + 1$
  4: $\mathsf{st}_\mathsf{B} \leftarrow \mathsf{st}$
  5: $\mathsf{tr}_\mathsf{B} \leftarrow \mathsf{tr}_\mathsf{B} \,||\, (\mathsf{ad}, \mathsf{ct})$
  6: $\mathsf{state}[\mathsf{B}, \mathsf{tr}_\mathsf{B}] \leftarrow \mathsf{st}_\mathsf{B}$
  7: $\mathsf{key}[\mathsf{B}, \mathsf{tr}_\mathsf{B}] \leftarrow \mathsf{k}$
  8: $\mathsf{log}[i] \leftarrow (\text{“rec”}, \mathsf{tr}_\mathsf{B}, \mathsf{ad}, \mathsf{ct})$
  9: return

Oracle EXP-STATE($\mathcal{P}$, tr)
  1: $i \leftarrow i + 1$
  2: $\mathsf{log}[i] \leftarrow (\text{“stexp”}, \mathcal{P}, \mathsf{tr})$
  3: return $\mathsf{state}[\mathcal{P}, \mathsf{tr}]$

Oracle EXP-KEY($\mathcal{P}$, tr)
  1: $i \leftarrow i + 1$
  2: $\mathsf{log}[i] \leftarrow (\text{“kexp”}, \mathcal{P}, \mathsf{tr})$
  3: return $\mathsf{key}[\mathcal{P}, \mathsf{tr}]$

Oracle LEAK-STATE($f$, $\mathcal{P}$, tr)
  1: $i \leftarrow i + 1$
  2: $\mathsf{log}[i] \leftarrow (\text{“stleak”}, f, \mathcal{P}, \mathsf{tr})$
  3: return $f(\mathsf{state}[\mathcal{P}, \mathsf{tr}])$

Oracle LEAK-KEY($f$, $\mathcal{P}$, tr)
  1: $i \leftarrow i + 1$
  2: $\mathsf{log}[i] \leftarrow (\text{“kleak”}, f, \mathcal{P}, \mathsf{tr})$
  3: return $f(\mathsf{key}[\mathcal{P}, \mathsf{tr}])$
```

Fig. 5: URKE oracles which use variables st_*, tr_*, state, key, log, and i that are all initialized in games where the oracles are used.

game, which returns an uniformly random bit if the predicates evaluates to true (see Appendix A.2 for a definition of $\mathbf{abort}_{\mathsf{IND}}$). We define trivial-KIND by using different sub-predicates, each of which indicates a trivial attack to exclude. The function inputs log and checks if any of the predicates evaluates to true. If it is the case the trivial-KIND function outputs true, thereby indicating that a trivial attack against the scheme happened, otherwise it outputs false. This allows to catch game runs where the adversary wins with a trivial attack. In what follows we explain how to define the sub-predicates that compose trivial-KIND.

Balli, Rösler and Vaudenay [BRV20a] define key indistinguishability for URKE *without* leakage resilience, i.e., in their game the adversary does not have access to LEAK-STATE and LEAK-KEY oracles. We recall the trivial attacks that arise without the leakage oracles (using a new, predicates-based formalization) and we then analyze the new class of trivial attacks that our additional leak oracles enable.

(P1): The adversary challenges a key and exposes it:

$$\exists\, i, j, \mathcal{P}, \mathsf{tr}\colon \mathsf{log}[j] = (\text{“kexp”}, \cdot, \mathcal{P}, \mathsf{tr}) \wedge\ \mathsf{log}[i] = (\text{“chall”}, \cdot, \mathsf{tr}, \cdot).$$

```
Game $\mathsf{LR\text{-}KIND}^{\ell}_{\mathcal{A}}(1^\lambda)$
1 : $b \leftarrow\!\!\$ \{0,1\}$
2 : $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$;  $(\mathsf{st_A}, \mathsf{st_B}, z) \leftarrow \mathsf{Init}(\mathsf{pp})$
3 : $\mathsf{state}[\cdot,\cdot], \mathsf{key}[\cdot,\cdot], \mathsf{log}[\cdot], \mathsf{tr_A}, \mathsf{tr_B} \leftarrow \perp$
4 : $\mathsf{state}[\mathsf{A}, \epsilon], \mathsf{state}[\mathsf{B}, \epsilon] \leftarrow (\mathsf{st_A}, \mathsf{st_B})$
5 : $i \leftarrow 0$
6 : $\mathsf{chall}[\cdot,\cdot] \leftarrow \mathsf{false}$
7 : $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, z)$
8 : if $\mathsf{trivial\text{-}KIND}(\mathsf{log}, \ell)$ then $\mathbf{abort}_{\mathsf{IND}}$
9 : return $1_{b=b'}$
```

```
Oracle $\mathsf{CHALL\text{-}KIND}(\mathcal{P}, \mathsf{tr})$
1 : $i \leftarrow i + 1$
2 : if $\mathsf{chall}[\mathcal{P}, \mathsf{tr}]$ :
3 :     return $\perp$
4 : $\mathsf{k} \leftarrow \mathsf{key}[\mathcal{P}, \mathsf{tr}]$
5 : if $\mathsf{key}[\mathcal{P}, \mathsf{tr}] \neq \perp \wedge b = 1$ :
6 :     $\mathsf{k} \leftarrow\!\!\$ \mathcal{K}$
7 : $\mathsf{chall}[\mathcal{P}, \mathsf{tr}] \leftarrow \mathsf{true}$
8 : $\mathsf{log}[i] \leftarrow (\text{“chall”}, \mathcal{P}, \mathsf{tr}, \perp)$
9 : return $\mathsf{k}$
```

Fig. 6: URKE LR-KIND game for the oracle set $\mathcal{O} = \{$SEND, RECEIVE, EXP-STATE, EXP-KEY, LEAK-STATE, LEAK-KEY, CHALL-KIND$\}$.

(P2): The adversary exposes the state of B and uses the exposed state to reproduce B's computations by using the Receive algorithm, thereby obtaining all the keys after the exposure:

$$\exists\, i, j, \mathsf{tr}, \mathsf{tr}^* : \mathsf{log}[i] = (\text{“stexp”}, \mathsf{B}, \mathsf{tr}) \wedge \mathsf{log}[j] = (\text{“chall”}, \cdot, \mathsf{tr}^*, \cdot) \wedge (\mathsf{tr} \prec \mathsf{tr}^*).$$

(P3): The adversary exposes the state of A and impersonates A by using the Send algorithm—blocking any possibility for A to heal with fresh randomness—and feeding the RECEIVE oracle with the resulting outputs, thereby bringing B out-of-sync and forcing B to retrieve a key that the adversary already knows:

$$\begin{aligned}\exists\, i, j, k, \mathsf{tr}, \mathsf{tr}^* : \ &\mathsf{log}[i] = (\text{“stexp”}, \mathsf{A}, \mathsf{tr}) \ \wedge \\ &\mathsf{log}[j] = (\text{“rec”}, \mathsf{B}, \mathsf{tr}^*, \cdot, \cdot) \wedge (\mathsf{tr} \prec \mathsf{tr}^*) \ \wedge \\ &\mathsf{log}[k] = (\text{“chall”}, \mathsf{B}, \mathsf{tr}^*, \cdot) \ \wedge \\ &[\nexists\, l, \mathsf{tr}' : (\mathsf{log}[l] = (\text{“send”}, \mathsf{A}, \mathsf{tr}', \cdot, \cdot, \perp) \wedge (\mathsf{tr} \prec \mathsf{tr}' \preceq \mathsf{tr}^*))].\end{aligned}$$

(P4): The adversary exposes the state of A and uses the exposed state to reproduce A's computation while using the Send algorithm only with manipulated randomness (which blocks any possibility for A to heal), thereby obtaining all the keys after the exposure.

$$\begin{aligned}\exists\, i, j, \mathsf{tr}, \mathsf{tr}^* : \ &\mathsf{log}[i] = (\text{“stexp”}, \mathsf{A}, \mathsf{tr}) \ \wedge \\ &\mathsf{log}[j] = (\text{“chall”}, \mathsf{A}, \mathsf{tr}^*, \cdot) \wedge (\mathsf{tr} \prec \mathsf{tr}^*) \ \wedge \\ &[\nexists\, l, \mathsf{tr}' : (\mathsf{log}[l] = (\text{“send”}, \mathsf{A}, \mathsf{tr}', \cdot, \cdot, \perp) \wedge (\mathsf{tr} \prec \mathsf{tr}' \preceq \mathsf{tr}^*))].\end{aligned}$$

When adding the two leakage oracles LEAK-STATE and LEAK-KEY to the game, a few more trivial attacks arise. Recall that the leakage function returns a single bit, as explained in Appendix A.2, and the leakage is bounded by ℓ.

In the LR-KIND game, any call to the LEAK-KEY oracle before receiving a challenge key could leak information about that key, so the corresponding leakage bound is set to zero. Leakage on B is unbounded only if no challenge key will be received after the leakage. Since unbounded possibility of leakage is equivalent to exposing the state (which is allowed if no challenge key is sent or received later), it is meaningless to continue to enforce leakage bounds at that moment.

Informally, these predicates enforce that the adversary cannot leak any information on the key that it challenges. Leaking more than the leakage bound on any key and on a state of the sending party A is not permitted. More precisely, every time the adversary calls the SEND oracle with fresh randomness, the adversary can leak up to ℓ before the next call to the SEND oracle with fresh randomness. Furthermore, between a challenge key send with manipulated randomness and the last unmanipulated send before it, the state of the sender should not be leaked, since this would allow to leak on the challenge key which is a deterministic function of all states of the sender since its last send with fresh randomness. Also, leakage on the receiver before the last challenge key was received is forbidden, since an adversary knowing all messages which the receiver will receive until the challenge can compute a bit of the challenge key by simulating the reception of these not yet received messages and the challenge ciphertext. The knowledge of future messages is possible as the protocol is asynchronous.

(P5): The adversary challenges a key that it previously leaked:

$$\exists\, i, j, \mathsf{tr}\colon \mathsf{log}[i] = (\text{“}\mathsf{kleak}\text{”}, \cdot, \mathsf{tr}) \wedge \mathsf{log}[j] = (\text{“}\mathsf{chall}\text{”}, \cdot, \mathsf{tr}, \cdot).$$

Intuition: Leaking a challenge key trivially allows to distinguish it from a random key (with probability $1 - \frac{1}{2^n}$ if n bits are leaked).

(P6): The adversary leaks more than ℓ bits of A's state, either at once or while manipulating the randomness that the SEND oracle uses in order to impersonate A, and then challenges on the key B derives:

$$\begin{aligned}&\exists\, i, j, \mathsf{tr}, \mathsf{tr}'\colon (\mathsf{tr} \preceq \mathsf{tr}') \wedge \\ &\quad \mathsf{log}[i] = (\text{“}\mathsf{rec}\text{”}, \mathsf{B}, \mathsf{tr}', \cdot, \cdot) \wedge \mathsf{log}[j] = (\text{“}\mathsf{chall}\text{”}, \mathsf{B}, \mathsf{tr}', \cdot) \wedge \\ &\quad [\nexists\, k, \mathsf{tr}_k\colon (\mathsf{tr} \prec \mathsf{tr}_k \preceq \mathsf{tr}') \wedge (\mathsf{log}[k] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}_k, \cdot, \cdot, \bot))] \wedge \\ &\quad |\{l, \mathsf{tr}_l\colon (\mathsf{log}[l] = (\text{“}\mathsf{stleak}\text{”}, \cdot, \mathsf{A}, \mathsf{tr}_l)) \wedge (\mathsf{tr} \preceq \mathsf{tr}_l \preceq \mathsf{tr}')\}| > \ell)\end{aligned}$$

Intuition: If there is no SEND between more than ℓ LEAK-STATE calls, the adversary can learn more than ℓ bits on a state of A. The same is true if there are SEND calls, but randomness used by them is manipulated. Then, it is a trivial attack if the adversary then impersonates A using leakage from this and challenges on the key output by B. Note challenging on keys derived by A is covered in the previous predicate.

(P7): The adversary leaks A's state and manipulates the randomness of SEND, thereby forbidding A from healing and making SEND deterministic, i.e., forcing A to produce a key that the adversary has learned a bit of:

$$\begin{aligned}\exists\, i, j, i', \mathsf{tr}, \mathsf{tr}' :\ & \mathsf{log}[i] = (\text{“}\mathsf{stleak}\text{”}, \mathsf{A}, \mathsf{tr}) \wedge \\ & \mathsf{log}[j] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}', \cdot, \cdot, \mathsf{r} \neq \bot) \wedge (\mathsf{tr} \prec \mathsf{tr}') \wedge \\ & \mathsf{log}[i'] = (\text{“}\mathsf{chall}\text{”}, \cdot, \mathsf{tr}', \cdot) \wedge \\ & [\nexists\, k, \mathsf{tr}^* : (\mathsf{tr} \prec \mathsf{tr}^* \prec \mathsf{tr}') \wedge \mathsf{log}[k] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}^*, \cdot, \cdot, \bot)].\end{aligned}$$

Intuition: The adversary can learn a bit of the challenge key, which is a deterministic function of A's state as long as the SEND oracle uses manipulated randomness, and can be computed by the leakage functions.

(P8): The adversary leaks B's state at any time before querying the $\mathsf{CHALL\text{-}KIND}$ oracle on any party (since A and B produce the same keys by correctness):

$$\exists\, i, j, \mathsf{tr}, \mathsf{tr}^* : \mathsf{log}[i] = (\text{“}\mathsf{stleak}\text{”}, \mathsf{B}, \mathsf{tr}) \wedge \mathsf{log}[j] = (\text{“}\mathsf{chall}\text{”}, \cdot, \mathsf{tr}^*, \cdot) \wedge (\mathsf{tr} \prec \mathsf{tr}^*).$$

Intuition: Since any exchanged key is a deterministic function of B's state and the messages that A sends to B, leaking B's state allows the decryption of a bit of all not yet received ciphertexts. If there was a challenge ciphertext among them, this would leak to a trivial win. This predicate is the equivalent of predicate P2 for leakage.

Optimality. These predicates (P1 to P8) are *optimal* within our formalism in which the adversary can freely choose any efficient function as a leakage function, in the sense any set of predicates which allows for more interactions cannot be realized by a correct construction any more. The first four predicates (P1 to P4) do not involve leakage and are the same that Balli, Rösler and Vaudenay identify as optimal for URKE under randomness manipulation [BRV20a], but for the sake of modularity we use a different formalism to define them. The remaining predicates (P5 to P8) deal with the new leakage oracles. The optimality of P5, P7 and P8 arises from the attacks described in the intuition paragraphs above, which directly give the adversary a trivial win, regardless of how and when the predicates are violated. Predicate P6 bounds the leakage of sender states, which forbids the adversary to gain enough information on A's state for an impersonation attack.

Definition 8. *A URKE scheme is (q, ℓ, t, ϵ)-$\mathsf{LR\text{-}KIND}$ secure for a security parameter $\lambda \in \mathbb{N}$, if we have, for leakage bound ℓ and for all adversaries $\mathcal{A}$ running in time at most t and making at most q oracle queries (including random oracle queries made by leakage functions) in the $\mathsf{LR\text{-}KIND}$ game in Fig. 6:*

$$2 \cdot \left| \frac{1}{2} - \Pr[\mathsf{LR\text{-}KIND}^{\ell}_{\mathcal{A}}(1^{\lambda}) \Rightarrow 1] \right| \leq \epsilon,$$

where the probability is taken over the random coins used in the game $\mathsf{LR\text{-}KIND}$ by challenger and adversary and all game runs, including aborted ones.

4.2 Construction

In Fig. 7 we present an LR-KIND-secure URKE that we construct from a bounded leakage-resilient MAC scheme and a classic kuKEM scheme. The construction from MAC and kuKEM is essentially the same as in [BRV20a] except that the MAC scheme in our construction is required to tolerate leakage of up to ℓ bits, where leakage only occurs on the Tag operation. This allows us to achieve LR-KIND security for the construction, as detailed below.

```
Function Setup$(1^\lambda)$
  $\mathsf{pp}_{\mathit{\Pi}_k} \leftarrow \mathit{\Pi}_k.\mathsf{Setup}(1^\lambda)$
  $\mathsf{pp} \leftarrow (\lambda, \mathsf{pp}_{\mathit{\Pi}_k})$
  return $\mathsf{pp}$

Function Send$(\mathsf{st_A}, \mathsf{ad})$
  $(\mathsf{pk}, k_s, k_m, \mathsf{tr}) \leftarrow \mathsf{st_A}$
  $(\mathsf{pk}, k_e, \mathsf{ct}') \leftarrow \mathit{\Pi}_k.\mathsf{Encaps}(\mathsf{pk})$
  $t \leftarrow \mathit{\Pi}_{lrm}.\mathsf{Tag}(k_m, (\mathsf{ad}, \mathsf{ct}'))$
  $\mathsf{ct} \leftarrow (\mathsf{ct}', t)$
  $\mathsf{tr} \leftarrow \mathsf{tr} \,||\, (\mathsf{ad}, \mathsf{ct})$
  $(k_o, k_s, k_m, u) \leftarrow H(k_e, k_s, \mathsf{tr})$
  $\mathsf{pk} \leftarrow \mathit{\Pi}_k.\mathsf{UpPk}(\mathsf{pk}, u)$
  $\mathsf{st_A} \leftarrow (\mathsf{pk}, k_s, k_m, \mathsf{tr})$
  return $\mathsf{st_A}, \mathsf{ct}, k_o$
```

```
Function Init$(\mathsf{pp})$
  $(\lambda, \mathsf{pp}_{\mathit{\Pi}_k}) \leftarrow \mathsf{pp};\ \mathcal{K} \leftarrow \mathsf{keyspace}(\lambda)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathit{\Pi}_k.\mathsf{Gen}(\mathsf{pp}_{\mathit{\Pi}_k})$
  $k_s, k_m \leftarrow\!\!\$\ \mathcal{K};\ \mathsf{tr} \leftarrow \bot$
  $\mathsf{st_A} \leftarrow (\mathsf{pk}, k_s, k_m, \mathsf{tr})$
  $\mathsf{st_B} \leftarrow (\mathsf{sk}, k_s, k_m, \mathsf{tr})$
  return $\mathsf{st_A}, \mathsf{st_B}$

Function Receive$(\mathsf{st_B}, \mathsf{ad}, \mathsf{ct})$
  $(\mathsf{sk}, k_s, k_m, \mathsf{tr}) \leftarrow \mathsf{st_B}$
  $(\mathsf{ct}', t) \leftarrow \mathsf{ct}$
  if $\neg\mathit{\Pi}_{lrm}.\mathsf{Ver}(k_m, (\mathsf{ad}, \mathsf{ct}'), t)$ then
    return $(\mathsf{false}, \bot, \bot)$
  $(\mathsf{sk}, k_e) \leftarrow \mathit{\Pi}_k.\mathsf{Decaps}(\mathsf{sk}, \mathsf{ct}')$
  if $k_e = \bot$ then return $(\mathsf{false}, \bot, \bot)$
  $\mathsf{tr} \leftarrow \mathsf{tr} \,||\, (\mathsf{ad}, \mathsf{ct}')$
  $(k_o, k_s, k_m, u) \leftarrow H(k_e, k_s, \mathsf{tr})$
  $\mathsf{sk} \leftarrow \mathit{\Pi}_k.\mathsf{UpSk}(\mathsf{sk}, u)$
  $\mathsf{st_B} \leftarrow (\mathsf{sk}, k_s, k_m, \mathsf{tr})$
  return $\mathsf{true}, \mathsf{st_B}, k_o$
```

Fig. 7: URKE construction from kuKEM $\mathit{\Pi}_k$ (Definition 4), MAC $\mathit{\Pi}_{lrm}$ (Definition 1) and function $H \colon \mathcal{K}^2 \times T \to \mathcal{K}^4$ where T is the set of all possible traces and $\mathcal{K}$ the set of keys, which is used by $\mathit{\Pi}_k$ and $\mathit{\Pi}_{lrm}$ as explained in Section 3. The same scheme, but with an LR-KUOWR-secure kuKEM scheme $\mathit{\Pi}_{lrk}$ (Definition 5) instead of the KUOWR-secure $\mathit{\Pi}_k$ (Definition 5) achieves LR-OW security as discussed in Section 5.

Correctness of our URKE follows from the correctness of the underlying kuKEM and MAC schemes. Namely, for a given consistent pair of queries to Send and Receive, by MAC correctness, the MAC verification call succeeds, and by kuKEM correctness, decapsulation outputs the key that was encapsulated. The trace tr is consistently updated in Send and $\mathsf{Receive}$, H is queried with the same input in both algorithms, and UpPk and UpSk are called with the same input. Thus, the same key k_o is output by Send and $\mathsf{Receive}$.

Theorem 1. *The construction that we define in Fig. 7 is an* $\mathsf{LR\text{-}KIND}$*-secure URKE scheme, assuming that the function H is modeled as a random oracle.*

More precisely, for leakage bound ℓ and security parameter λ (which gives $\mathcal{K} \leftarrow \mathsf{keyspace}(\lambda)$) and time t, $\tilde{t}$ and t' such that $t' \approx t \approx \tilde{t}$, and three natural numbers q_s, q_h and q with $q \geq q_s + q_h$, assume that there is some $\epsilon_{\mathsf{KUOWR}}$ such that the kuKEM scheme that the construction uses is $(2q, t', \epsilon_{\mathsf{KUOWR}})$-$\mathsf{KUOWR}$*-secure and some $\epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}$ such that the MAC scheme is $(q, \ell, \tilde{t}, \epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}})$-*$\mathsf{LR\text{-}OT\text{-}SUF}$ *secure. Then for any adversary $\mathcal{A}$ respecting leakage bound ℓ, running in time at most t and making at most q oracle queries in total, out of which q_h are random oracle queries and at most q_s are calls to either* SEND *or* $\mathsf{RECEIVE}$*, the success probability in the* $\mathsf{LR\text{-}KIND}$ *game for the construction from Fig. 7 with parameter λ is at most $\epsilon^{(q_s,q_h)}_{\mathsf{LR\text{-}KIND}}$ where*

$$\epsilon^{(q_s,q_h)}_{\mathsf{LR\text{-}KIND}} = \frac{(4(q_h + q_s))^2}{|\mathcal{K}|} + \epsilon_{\mathsf{KUOWR}} + \frac{q_h}{|\mathcal{K}|} + q_s \epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}.$$

Therefore, the construction is $(q, \ell, t, \epsilon^{(q_s,q_h)}_{\mathsf{LR\text{-}KIND}})$-$\mathsf{LR\text{-}KIND}$*-secure.*

Proof. We sketch the proof here and defer the details to Appendix C. The proof strategy is similar to the one used by Balli, Rösler and Vaudenay to prove security of their URKE scheme [BRV20b, Appendix B].

After excluding any collision in the random oracle, we show that an adversary can learn a challenge key only by querying the random oracle with inputs that lead the random oracle to output the challenge key itself. Based on this observation, we distinguish three cases depending on whether the challenge key the adversary obtains from the random oracle was obtained by (1) a call to SEND with fresh randomness ($\mathsf{r} = \perp$), (2) a call to SEND with manipulated randomness ($\mathsf{r} \neq \perp$), or (3) only by querying $\mathsf{RECEIVE}$ but not SEND. The first case reduces to the KUOWR security of the kuKEM scheme, the second to guessing a random key and the third reduces to the $\mathsf{LR\text{-}OT\text{-}SUF}$ security of the MAC scheme. The reduction in the second case is due to P7 and P4. These forbit to leak on (or expose) the state key k_s between initialization or the last non-manipulated SEND call and the challenge key SEND, and therefore force the adversary to guess the state key k_s in order to have the full input to the random oracle. □

Remark 1. In the construction of Fig. 7, the states $\mathsf{st_A}$ and $\mathsf{st_B}$ grow with each update since both Send and $\mathsf{Receive}$ append the current $(\mathsf{ad}, \mathsf{ct})$ pair to the trace, which is in turn stored in the respective states (lines 5 and 8 in Send and lines 7 and 10 in $\mathsf{Receive}$). We can avoid this by *not* storing tr in the states and use $\mathsf{tr} = (\mathsf{ad}, \mathsf{ct})$ as input to H, that is, $H(k_e, k_s, (\mathsf{ad}, \mathsf{ct}))$ on line 6 of Send and line 8 of $\mathsf{Receive}$, since H accumulates the entire transcript into k_s. One can invoke the collision-resistance of H as a random oracle to prove the security of this optimization.

Remark 2. A deep practical analysis of the expected overhead of achieving leakage resilience is left for future work, but we can already give an estimate for the above LR-KIND-secure construction, which uses an LR-OT-SUF-secure MAC scheme. We argued in Section 3.1 that LR-OT-SUF-secure MAC can be built in the random oracle model. In the standard model, a concrete instantiation was proposed by Hazay et al. [Haz+16, Section 5.3]. Their construction requires a (weak) hash proof system (e.g., as efficient as the Cramer-Shoup cryptosystem [CS98]), a (cheap) information-theoretically-secure one-time MAC and a leakage-resilient MAC secure under a notion where no verification queries are allowed [Haz+16, Definition 5.2] where tags are of size $O(\lambda)$ pseudorandom function outputs, where each function has a superpolynomial output of length $O(\lambda^{\omega(1)})$, for security parameter λ.

Remark 3. In Appendix B we describe why the URKE scheme of [BRV20a] is not in general secure under our security notions. Our attack exploits the fact that if the underlying MAC is not leakage-resilient, then leaking enough bits of the MAC secret allows the adversary to impersonate towards B and trivially win a challenge on the key derived by B.

5 Leakage-resilient one-wayness for URKE

The predicate trivial-KIND in the LR-KIND game that we introduce in Section 4.1 is very strong, i.e., it strongly limits the oracle calls that the adversary can perform in the game. This means that LR-KIND security can only be achieved for very weak adversaries. In particular, a model allowing no leakage on the receiver seems not very practical, as side channels on the receiver can exist.

In this section we introduce a stronger security notion that enables the adversary to leak more information, especially on the receiver. We refer to this notion as one-wayness for unidirectional ratcheted key exchange under leakage (LR-OW). We first define LR-OW and we then present a construction for an LR-OW-secure URKE from an LR-KUOWR-secure kuKEM (Definition 5) and an LR-OT-SUF-secure MAC scheme (Definition 3). While LR-OT-SUF MAC-schemes are known as described in the above section, LR-KUOWR-secure kuKEM was never required before and we therefore construct it in Section 6.1.

5.1 Security definition

The LR-OW game for one-wayness under leakage is given in Fig. 8. It relies on the same set of oracles as LR-KIND (Fig. 5), but it uses CHALL-OW instead of CHALL-KIND.

In the LR-OW game with leakage bounds ℓ_{A} for leakage on A, ℓ_{B} for leakage on B and ℓ_{k} for leakage on exchanged keys, the function $\mathsf{trivial\text{-}OW}(\mathsf{log}, \ell_{\mathsf{A}}, \ell_{\mathsf{B}}, \ell_{\mathsf{k}})$ plays the same role as trivial-KIND but for the LR-OW game. We define trivial-OW by using different sub-predicates; the function returns true if at least one of the predicates evaluates to true. The trivial-OW function is a disjunctive clause of

```
Game $\mathsf{LR\text{-}OW}_{\mathcal{A}}^{\ell_\mathsf{A},\ell_\mathsf{B},\ell_\mathsf{k}}(1^\lambda)$
1 :  $b \leftarrow\!\!\$\ \{0,1\}$
2 :  $\mathsf{win} \leftarrow \mathsf{false}$
3 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
4 :  $(\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B}, z) \leftarrow \mathsf{Init}(\mathsf{pp})$
5 :  $\mathsf{state}[\cdot,\cdot], \mathsf{key}[\cdot,\cdot], \mathsf{log}[\cdot], \mathsf{tr}_\mathsf{A}, \mathsf{tr}_\mathsf{B} \leftarrow \bot$
6 :  $\mathsf{state}[\mathsf{A},\epsilon], \mathsf{state}[\mathsf{B},\epsilon] \leftarrow (\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B})$
7 :  $i \leftarrow 0$
8 :  $\mathcal{A}^{\mathcal{O}}(\mathsf{pp}, z)$
9 :  if $\mathsf{trivial\text{-}OW}(\mathsf{log}, \ell_\mathsf{A}, \ell_\mathsf{B}, \ell_\mathsf{k})$ then
10 :   abort
11 : return $\mathsf{win}$
```

```
Oracle $\mathsf{CHALL\text{-}OW}(\mathcal{P}, \mathsf{tr}, k)$
1 :  $i \leftarrow i + 1$
2 :  if $\mathsf{key}[\mathcal{P},\mathsf{tr}] \neq \bot \wedge \mathsf{key}[\mathcal{P},\mathsf{tr}] = k$ :
3 :  $\mathsf{win} \leftarrow \mathsf{true}$
4 :  $\mathsf{log}[i] \leftarrow (\text{“}\mathsf{chall}\text{”}, \mathcal{P}, \mathsf{tr}, k)$
5 :  return $\mathsf{win}$
```

Fig. 8: URKE $\mathsf{LR\text{-}OW}$ game for the oracle set $\mathcal{O} = \{\mathsf{SEND}, \mathsf{RECEIVE}, \mathsf{EXP\text{-}STATE}, \mathsf{EXP\text{-}KEY}, \mathsf{LEAK\text{-}STATE}, \mathsf{LEAK\text{-}KEY}, \mathsf{CHALL\text{-}OW}\}$.

the predicates (P1), (P2), (P3), (P4) that we define in Section 4.1 together with the predicate that we define next (P9 to P11).

(P9): The adversary leaks more than ℓ_A bits on a state of A or a sequence of states of A between which randomness is continuously manipulated:

$$\begin{aligned}&\exists\, i, j, \mathsf{tr}, \mathsf{tr}' \colon (\mathsf{tr} \preceq \mathsf{tr}') \wedge \\ &\quad \mathsf{log}[i] = (\text{“}\mathsf{rec}\text{”}, \mathsf{B}, \mathsf{tr}', \cdot, \cdot) \wedge \mathsf{log}[j] = (\text{“}\mathsf{chall}\text{”}, \mathsf{B}, \mathsf{tr}', \cdot) \wedge \\ &\quad [\nexists\, k, \mathsf{tr}_k \colon (\mathsf{tr} \prec \mathsf{tr}_k \preceq \mathsf{tr}') \wedge (\mathsf{log}[k] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}_k, \cdot, \cdot, \bot))] \wedge \\ &\quad |\{l, \mathsf{tr}_l \colon (\mathsf{log}[l] = (\text{“}\mathsf{stleak}\text{”}, \cdot, \mathsf{A}, \mathsf{tr}_l)) \wedge (\mathsf{tr} \preceq \mathsf{tr}_l \preceq \mathsf{tr}')\}| > \ell_\mathsf{A})\end{aligned}$$

Intuition: See the intuition for (P6).

(P10): The adversary leaks more than ℓ_k bits of a challenge key:

$$\exists\, \mathsf{tr}, \mathcal{P}, j \colon \mathsf{log}[j] = (\text{“}\mathsf{chall}\text{”}, \mathcal{P}, \mathsf{tr}, \cdot) \wedge |\{i \colon \mathsf{log}[i] = (\text{“}\mathsf{kleak}\text{”}, \cdot, \mathcal{P}, \mathsf{tr},)\}| \geq \ell_\mathsf{k}.$$

Intuition: This predicate enforces the bound ℓ_k on the leakage on a challenge key. It depends on the scheme that instantiates the primitive, as ℓ_k varies with the parameters of the scheme.

(P11): The adversary leaks more than ℓ_B bits on B's state and then queries the $\mathsf{CHALL\text{-}OW}$ oracle:

$$\begin{aligned}&\exists\, \mathsf{tr}, j \colon |\{\mathsf{tr}_i, i \colon \mathsf{log}[i] = (\text{“}\mathsf{stleak}\text{”}, \cdot, \mathsf{B}, \mathsf{tr}_i,) \wedge (\mathsf{tr}_i \preceq \mathsf{tr})\}| \geq \ell_\mathsf{B} \wedge \\ &\qquad \mathsf{log}[j] = (\text{“}\mathsf{chall}\text{”}, \mathsf{B}, \mathsf{tr}, \cdot)\end{aligned}$$

Intuition: The adversary leaks more than the prescribed ℓ_{B} bits on B's state, which might enable the adversary to break the security of the scheme and the adversary queries the challenge oracle. This predicate depends on the scheme that instantiates the primitive.

The predicates that involve leakage are less restrictive for one-wayness than for key indistinguishability. In particular, bounded leakage on the receiver, some leakage prior to send calls with manipulated randomness and bounded leakage on challenge keys are allowed only for one-wayness. Through them, the adversary can learn up to $\ell_{\mathsf{A}} + \ell_{\mathsf{B}} + \ell_{\mathsf{k}}$ bits of the challenge key. All three predicates only enforce these bound and the bound on leakage on A's state which is required to exclude impersonation attacks (as in P6). The value of the bounds and the impact their violations have depend on the concrete instantiation of the primitive.

Definition 9. *A URKE scheme is* $(q, \ell_{\mathsf{A}}, \ell_{\mathsf{B}}, \ell_{\mathsf{k}}, t, \epsilon)$-$\mathsf{LR\text{-}OW}$ *secure for a fixed security parameter* λ*, if we have, for leakage bounds* $\ell_{\mathsf{A}}, \ell_{\mathsf{B}}, \ell_{\mathsf{k}}$ *and for all adversaries* $\mathcal{A}$ *running in time at most* t *(including runtime of leakage functions) and making at most* q *oracle queries (including random oracle queries, even those in leakage functions) in the* $\mathsf{LR\text{-}OW}$ *game in Fig. 8:*

$$\Pr[\mathsf{LR\text{-}OW}_{\mathcal{A}}^{\ell_{\mathsf{A}}, \ell_{\mathsf{B}}, \ell_{\mathsf{k}}}(1^{\lambda}) \Rightarrow 1] \leq \epsilon,$$

where the probability is taken over the random coins used in the game $\mathsf{LR\text{-}OW}$ *by challenger and adversary, and all game runs (even aborted ones).*

Remark 4. The $\mathsf{LR\text{-}OW}$ and $\mathsf{LR\text{-}KIND}$ URKE security notions are by themselves incomparable. More precisely, LR-OW asks the adversary to gain more knowledge about output keys, while LR-KIND requires it to operate with less leakage (so it has less information about output keys). Neither one of the notions directly implies the other:

1. If an URKE is such that even bounded leakage on the receiver allows the adversary to simulate enough of $\mathsf{Receive}$ to derive a key, then it is not $\mathsf{LR\text{-}OW}$-secure, but it may still be $\mathsf{LR\text{-}KIND}$ secure.
2. Consider a LR-OW URKE U. Then transform it into another URKE U' by appending one 0 bit to all exchanged keys k (and use $\{0,1\} \times \mathcal{K}$ as keyspace for exchanged keys). Then U' is still $\mathsf{LR\text{-}OW}$-secure but clearly not $\mathsf{LR\text{-}KIND}$.

Remark 5. We believe that in the ROM, any $\mathsf{LR\text{-}OW}$-secure URKE U can likely be transformed into an $\mathsf{LR\text{-}KIND}$-secure URKE U' by defining the keys exchanged via U' to be the output of the random oracle on those obtained by U. This builds on the intuition that $\mathsf{LR\text{-}OW}$ allows for more leakage and is therefore in some sense "stronger" than $\mathsf{LR\text{-}KIND}$ security, even if the two notions are formally incomparable as detailed above; we leave formalizing this to future work.

5.2 Construction

An LR-OW-secure URKE can be constructed from a kuKEM and a MAC in exactly the same way than an LR-KIND-secure URKE, as depicted in Fig. 7 (page 19). The only difference is that the kuKEM (Π_k in the scheme of Fig. 7) now is replaced by an LR-KUOWR-secure one Π_{lrk} (Definition 5) which tolerates ℓ_B bits of leakage on the receiver before a challenge key reception. Furthermore, the MAC scheme needs to be LR-OT-SUF secure for the bound $\ell_\mathsf{A} + \ell_\mathsf{B}$ in the LR-OW-secure construction, and not only for bound ℓ_A.

Since an LR-KUOWR-secure kuKEM is also KUOWR-secure, this construction is a special case of our LR-KIND-secure construction and therefore LR-KIND- and LR-OW-secure.

Theorem 2. *The construction of Fig. 7 defines an* LR-OW*-secure URKE scheme, where we model the function H as a random oracle.*

*More precisely, for leakage bounds $\ell_\mathsf{A}, \ell_\mathsf{B}, \ell_\mathsf{k}$ and security parameter λ (which induces $\mathcal{K} = \mathsf{keyspace}(\lambda)$) and times t, t' and $\tilde{t}$ such that $t' \approx t \approx \tilde{t}$, and three natural numbers q_s, q_h and q such that $q \geq q_h + q_s$, assume that there is some $\epsilon_{\mathsf{LR\text{-}KUOWR}}$ such that the kuKEM scheme that the construction uses is $(2q + 1, \ell_\mathsf{B}, t', \epsilon_{\mathsf{LR\text{-}KUOWR}})$-*LR-KUOWR*-secure and some $\epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}$ such that the MAC scheme is $(q, \ell_\mathsf{A} + \ell_\mathsf{B}, \tilde{t}, \epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}})$-*LR-OT-SUF *secure. Then any adversary $\mathcal{A}$ running in time at most t (including its leakage functions) and making at most q oracles queries, out of which at most q_h are random oracle queries (including those made by leakage functions) and at most q_s are calls to either* SEND *or* RECEIVE*, has a success probability upper bounded by $\epsilon^{(q_s,q_h)}_{\mathsf{LR\text{-}OW}}$ in the* LR-OW *game against the construction of Fig. 7, where*

$$\epsilon^{(q_s,q_h)}_{\mathsf{LR\text{-}OW}} = \frac{2^{\ell_\mathsf{A}+\ell_\mathsf{B}+\ell_\mathsf{k}}}{|\mathcal{K}|} + \frac{(4(q_h+q_s))^2}{|\mathcal{K}|} + \epsilon_{\mathsf{LR\text{-}KUOWR}} + q_h \frac{2^{\ell_\mathsf{A}+\ell_\mathsf{B}}}{|\mathcal{K}|} + q_s \epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}.$$

*As a consequence, the construction is $(q, \ell_\mathsf{A}, \ell_\mathsf{B}, \ell_\mathsf{k}, t, \epsilon^{(q,q)}_{\mathsf{LR\text{-}OW}})$-*LR-OW *secure.*

Proof. We provide a proof sketch here and defer the full proof to Appendix D We adapt the proof for the LR-KIND-secure construction (Theorem 1) to the LR-OW setting. The main difference is that the adversary can also win the LR-OW game without querying the random oracle, by guessing the challenge key among all keys which match the leakage on the challenge key. Furthermore, in the treatment of the case where the adversary wins after having obtained a challenge key as output of the random oracle, a few small changes occur in the reductions for the three subcases that we identify for the LR-KIND-security proof (Theorem 1). The first case—a call to SEND with fresh randomness—is now reduced to LR-KUOWR-security of the kuKEM, and for the other two cases, the leakage bounds and some oracle simulations slightly change. In particular, for case 2, P9 now replaces P7. Due to P9 and P11, the adversary can know at most $\ell_\mathsf{A} + \ell_\mathsf{B}$ bits of k_s, but it still has to guess the rest of it. □

6 On the complexity of leakage-resilient kuKEM

In this section, we first show that $\mathsf{LR\text{-}KUOWR}$-secure kuKEM can be constructed from leakage-resilient OW-CPA-secure hierarchical identity-based encryption and non-interactive zero-knowledge proofs. We then show that $\mathsf{LR\text{-}KUOWR}$-secure kuKEM can be constructed from $\mathsf{LR\text{-}OW}$-secure URKE and an $\mathsf{LR\text{-}OT\text{-}SUF}$-secure MAC. This latter result implies, given our $\mathsf{LR\text{-}OW}$-secure URKE construction (Sections 4 and 5), that relative to a random oracle and a $\mathsf{LR\text{-}OT\text{-}SUF}$-secure MAC, $\mathsf{LR\text{-}OW}$-secure URKE and $\mathsf{LR\text{-}KUOWR}$-secure kuKEM are in a meaningful sense *equivalent*. We then contextualise these results by discussing relations between different kuKEM and URKE variants.

6.1 Building $\mathsf{LR\text{-}KUOWR}$ kuKEM from $\mathsf{LR\text{-}OW\text{-}CPA}$ HIBE

Hierarchical identity-based encryption. We first recall appropriate definitions for hierarchical identity-based encryption, or HIBE.

Definition 10 (HIBE). *A* hierarchical identity-based encryption *(HIBE) comprises the following efficient algorithms:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{mk})$ *takes a unary string* 1^λ *and outputs public parameters* pp *and a master key* mk*. We assume* pp *is implicitly input to the algorithms below.*
- $\mathsf{Gen}(\vec{id}, \mathsf{mk}) \to \mathsf{sk}$ *takes an ID vector* $\vec{id}$ *and a master key* mk *and outputs a secret key* sk *associated with* $\vec{id}$*.*
- $\mathsf{Del}(\vec{id}, \mathsf{sk}_{\vec{id}}, id) \to \mathsf{sk}_{\vec{id}||id}$ *takes a vector of IDs* $\vec{id}$ *associated with secret key* $\mathsf{sk}_{\vec{id}}$ *and an ID* id *and outputs a secret key* $\mathsf{sk}_{\vec{id}||id}$ *associated with vector* $\vec{id} \;||\; id$
- $\mathsf{Enc}(m, \vec{id}) \to \mathsf{ct}$ *takes a message* m *and a vector of IDs* $\vec{id}$ *and outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) \to m$ *takes a ciphertext* ct *and secret key* sk *and outputs a message, where* $m = \perp$ *denotes failure.*

When convenient, we sometimes abuse notation and consider IDs as ID vectors.

Correctness. Intuitively, a HIBE is correct if for a given call $(\mathsf{pp}, \mathsf{mk}) \leftarrow \mathsf{Setup}(1^\lambda)$, if a message m is encrypted with respect to $\vec{id}$, i.e., $\mathsf{ct} \leftarrow \mathsf{Enc}(m, \vec{id})$, then any secret key sk associated with a vector of IDs that is a prefix or equal to $\vec{id}$ (derived from an appropriate calls to Gen and Del) should be such that $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) = m$. We provide a formal definition in Appendix E.1.

Security. We consider one-wayness notions under leakage where the goal of the adversary is to decrypt a randomly-chosen message encrypted under an identity for which they do not trivially know a secret key. Here, we consider notions with and without the aid of a decryption oracle ($\mathsf{LR\text{-}OW\text{-}CCA}$ and $\mathsf{LR\text{-}OW\text{-}CPA}$, respectively). Following [LRW11], our security notions consider two leakage bounds

```
Game $\mathsf{HIBE\text{-}X}_{\mathcal{A}}^{\ell_{\mathsf{mk}},\ell_{\mathsf{sk}}}(1^\lambda)$
1 : $(\mathsf{pp}, \mathsf{mk}) \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $m^* \leftarrow \mathcal{M}$
3 : $b^* \leftarrow \{0,1\}$
4 : $q_{\mathsf{mk}}, q_{\mathsf{sk}} \leftarrow 0$
5 : $\mathsf{sks}[\cdot] \leftarrow \perp$
6 : $\mathsf{chall} \leftarrow \mathsf{false}$;  $cid \leftarrow \perp$
7 : $out \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{ct}^*)$
8 : if $m^* = out$ then return $1$
9 : else return $0$

$\mathsf{CREATE}(\vec{id})$
1 : if $\mathsf{sks}[\vec{id}] \neq \perp$ then abort
2 : $\mathsf{sks}[\vec{id}] \leftarrow \mathsf{Gen}(\vec{id}, \mathsf{mk})$
3 : return $\mathsf{sks}[\vec{id}]$

$\mathsf{EXP\text{-}KEY}(\vec{id})$
1 : $XP \leftarrow XP \cup \{id : id \preceq \vec{id}\}$
2 : if $cid \in XP$ then abort
3 : return $\mathsf{sks}[\vec{id}]$
```

```
$\mathsf{DEC}(\mathsf{ct}, \vec{id})$
1 : if $\mathsf{ct} = \mathsf{ct}^*$ then abort
2 : return $\mathsf{Dec}(\mathsf{ct}, \mathsf{mk})$

$\mathsf{CHALL\text{-}OW}(\vec{id})$
1 : if $\mathsf{chall} = \mathsf{true}$ then abort
2 : if $\vec{id} \in XP$ then abort
3 : $\mathsf{chall} \leftarrow \mathsf{true}$
4 : $cid \leftarrow \vec{id}$
5 : $\mathsf{ct}^* \leftarrow \mathsf{Enc}(m^*, \vec{id})$
6 : return $\mathsf{ct}^*$

$\mathsf{LEAK}(\vec{id}, f)$
1 : if $\vec{id} = \perp$
2 :   if $q_{\mathsf{mk}} > \ell_{\mathsf{mk}}$ then abort
3 :   $q_{\mathsf{mk}} \leftarrow q_{\mathsf{mk}} + 1$
4 :   return $f(\mathsf{mk})$
5 : else
6 :   if $q_{\mathsf{sk}} > \ell_{\mathsf{sk}}$ then abort
7 :   $q_{\mathsf{sk}} \leftarrow q_{\mathsf{sk}} + 1$
8 : return $f(\mathsf{sks}[\vec{id}])$
```

Fig. 9: HIBE-X HIBE security notion for $\mathsf{HIBE\text{-}X} \in \{\mathsf{LR\text{-}OW\text{-}CPA}, \mathsf{LR\text{-}OW\text{-}CCA}\}$. Let $\mathcal{O}_{core} = (\mathsf{CREATE}, \mathsf{EXP\text{-}KEY})$. Then the oracle collection is $\mathcal{O} = \mathcal{O}_{\mathsf{core}} \cup \{\mathsf{LEAK}, \mathsf{CHALL\text{-}OW}\}$ for LR-OW-CPA and $\mathcal{O} = \mathcal{O}_{\mathsf{core}} \cup \{\mathsf{LEAK}, \mathsf{CHALL\text{-}OW}, \mathsf{DEC}\}$ for LR-OW-CCA.

with respect to each master key (ℓ_{mk}) and for all associated secret keys that descend from a given master key (ℓ_{sk}), which, looking ahead, suffices to build LR-KUOWR kuKEM. The notions are formally captured in Figure 9.

Definition 11 (LR-OW-CPA). *We say that an HIBE Π is $(q, \ell_{\mathsf{mk}}, \ell_{\mathsf{sk}}, t, \epsilon)$-LR-OW-CPA secure for security parameter λ if, for leakage bounds ℓ_{mk} and ℓ_{sk} and all adversaries $\mathcal{A}$ which make at most q oracle queries and run in time at most t, we have:*

$$\Pr[\mathsf{LR\text{-}OW\text{-}CPA}_{\mathcal{A}}^{\ell_{\mathsf{mk}},\ell_{\mathsf{sk}}}(1^\lambda) \Rightarrow 1] \leq \epsilon,$$

where the probability is taken over all the random coins that the challenger and the adversary use and the game LR-OW-CPA *is defined in Fig. 9 for $\mathcal{O} = (\mathsf{CREATE}, \mathsf{LEAK}, \mathsf{EXP\text{-}KEY}, \mathsf{CHALL\text{-}OW})$.*

Definition 12 (LR-OW-CCA). *We say that an HIBE $\mathit{\Pi}$ is $(q, \ell_{\mathsf{mk}}, \ell_{\mathsf{sk}}, t, \epsilon)$-* LR-OW-CCA *secure for security parameter λ if, for leakage bounds ℓ_{mk} and ℓ_{sk} and all adversaries $\mathcal{A}$ which make at most q oracle queries and run in time at most t, we have:*

$$\Pr[\mathsf{LR\text{-}OW\text{-}CCA}_{\mathcal{A}}^{\ell_{\mathsf{mk}},\ell_{\mathsf{sk}}}(1^{\lambda}) \Rightarrow 1] \leq \epsilon,$$

where the probability is taken over all the random coins that the challenger and the adversary use and the game LR-OW-CCA *is defined in Fig. 9 for $\mathcal{O} = (\mathsf{CREATE}, \mathsf{LEAK}, \mathsf{EXP\text{-}KEY}, \mathsf{CHALL\text{-}OW}, \mathsf{DEC})$.*

LR-OW-CCA-*secure HIBE from* LR-OW-CPA-*secure HIBE and NIZKs.* In Fig. 10 we construct LR-OW-CCA-secure HIBE from an LR-OW-CPA-secure HIBE scheme $\mathit{\Pi}_h$ and $\mathit{\Pi}_n$, a NIZK that satisfies completeness, composable zero-knowledge and true simulation f-extractability, as defined in Appendix E.2. The construction is essentially the same as that of the work of Dodis et al. [Dod+10c], that constructs LR-IND-CCA-secure public-key encryption from LR-IND-CPA-secure public-key encryption and NIZKs, but adapted to the syntax of HIBE. The construction indeed works even when considering one-way, rather than indistinguishability-based security notions. Intuitively, the NIZK ensures that the adversary must 'know' the message when making a decryption query in order for the call to succeed, and so the decryption oracle does not provide any 'useful' information and thus CPA security suffices. Correctness follows from the correctness of $\mathit{\Pi}_h$ and the completeness of $\mathit{\Pi}_n$.

```
Function Setup($1^\lambda$)
  $(\mathsf{pp}, \mathsf{mk}) \leftarrow \mathit{\Pi}_h.\mathsf{Setup}(1^\lambda)$
  $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathit{\Pi}_n.\mathsf{Setup}(1^\lambda)$
  $\mathsf{pp}' \leftarrow (\mathsf{pp}, \mathsf{crs})$
  return $(\mathsf{pp}, \mathsf{mk})$

Function Gen($\vec{id}, \mathsf{mk}$)
  return $\mathit{\Pi}_h.\mathsf{Gen}(\vec{id}, \mathsf{mk.mk})$

Function Del($\vec{id}, \mathsf{sk}_{\vec{id}}, id$)
  return $\mathit{\Pi}_h.\mathsf{Del}(\vec{id}, \mathsf{sk}_{\vec{id}}, id)$
```

```
Function Enc($\mathsf{pp}, m, \vec{id}; r$)
  $\mathsf{ct}' \leftarrow \mathit{\Pi}_h.\mathsf{Enc}(m, \vec{id}; r)$
  $\pi \leftarrow \mathit{\Pi}_n.\mathsf{Prove}(\mathsf{pp.crs}, ((m, r), (\vec{id}, \mathsf{ct})))$
  return $(\mathsf{ct}', \pi, \vec{id})$

Function Dec($\mathsf{pp}, \mathsf{ct}, \mathsf{sk}$)
  $\mathsf{ct} \leftarrow (\mathsf{ct}', \pi, \vec{id})$
  if $\mathit{\Pi}_n.\mathsf{Ver}(\mathsf{pp.crs}, (\vec{id}, \mathsf{ct}), \pi) = \mathsf{false}$ :
    return $\perp$
  return $\mathit{\Pi}_h.\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$
```

Fig. 10: Construction $\mathit{\Pi}^*$ of LR-OW-CCA-secure HIBE from LR-OW-CPA-secure HIBE $\mathit{\Pi}_h$ and complete, CZK-secure and f-TSE-secure NIZK $\mathit{\Pi}_n$ for relation $R = \{(x, y) = ((m, r), (\vec{id}, \mathsf{ct})) : c \leftarrow \mathsf{Enc}(\mathsf{pp}, m, \vec{id}; r)\}$. Functions other than Gen implicitly take pp as input and provide pp.pp as input to the corresponding function from $\mathit{\Pi}_h$.

Theorem 3. *Let Π_h be a $(q, \ell_{\mathsf{mk}}, \ell_{\mathsf{sk}}, t, \epsilon_{cpa})$-*$\mathsf{LR\text{-}OW\text{-}CPA}$*-secure HIBE, and Π_n a NIZK for relation $R = \{(x, y) = ((m, r), (\vec{id}, \mathsf{ct})) : c \leftarrow \mathsf{Enc}(\mathsf{pp}, m, \vec{id}; r)\}$ that is* complete, *(t, ϵ_z)-*CZK*-secure, and $(1, t, \epsilon_f)$-f-*TSE*-secure for $f(m, r) = m$. Then Π^* (Fig. 10) is $(q, \ell_{\mathsf{mk}}, \ell_{\mathsf{sk}}, t', \epsilon_{cpa} + \epsilon_c + \epsilon_f)$-*$\mathsf{LR\text{-}OW\text{-}CCA}$*-secure where $t \approx t'$.*

Proof. The proof can be found in Appendix E.3.

From HIBE to kuKEM. We provide a construction of $\mathsf{LR\text{-}KUOWR}$-secure kuKEM from $\mathsf{LR\text{-}OW\text{-}CCA}$-secure HIBE in Appendix E.4, which is the canonical construction of kuKEM from HIBE (in particular using Enc to implement Encaps, Del to implement UpSk, and Dec and Del to implement Decaps) following [PR18; BRV20a]. Although we do not provide a construction of $\mathsf{LR\text{-}KUOWR}$-secure kuKEM, we believe that one is in reach via a construction of $\mathsf{LR\text{-}OW\text{-}CPA}$-secure HIBE, since 1) leakage-resilient HIBE has been constructed before (see [LRW11]), and 2) unpredictability notions have been considered with leakage resilience before (signatures, one-way functions, etc.).

6.2 Building $\mathsf{LR\text{-}KUOWR}$ kuKEM from $\mathsf{LR\text{-}OW}$ URKE

In Sections 4 and 5, we showed that $\mathsf{LR\text{-}OW}$-secure URKE can be constructed from $\mathsf{LR\text{-}KUOWR}$-secure kuKEM and $\mathsf{LR\text{-}OT\text{-}SUF}$-secure MAC with polynomial security loss, except for additive terms of the form $2^{\ell}/|\mathcal{K}|$ for leakage bound ℓ, which due to the one-way nature of $\mathsf{LR\text{-}OW}$ game are unavoidable (since the adversary can always leak ℓ bits of the challenge key). We argue here that $\mathsf{LR\text{-}KUOWR}$-secure kuKEM can be built from $\mathsf{LR\text{-}OW}$-secure URKE and $\mathsf{LR\text{-}OT\text{-}SUF}$-secure MAC in the random oracle model (with polynomial loss).

Construction. Balli, Rösler and Vaudenay [BRV20a] showed that KUOWR-secure kuKEM can be built from $\mathsf{OT\text{-}SUF}$-secure MAC and a KIND-secure URKE. We modify their construction in two ways to build $\mathsf{LR\text{-}KUOWR}$-secure kuKEM and we present our resulting construction in Fig. 11. First, we now require leakage-resilient building blocks. Second, in [BRV20a], Send would directly output keys (k, k_m), but in our case since we do not rely on the key indistinguishability of the URKE but only one-wayness, the MAC key k_m may be very far from uniform. Thus, we feed the output key of Send through a random oracle H to derive k and k_m.

In more detail, a public key consists of URKE sender state $\mathsf{st_A}$, and the corresponding secret key of both the sender and receivers' states. At a high level, Encaps uses Send and Decaps uses $\mathsf{Receive}$, and UpPk and UpSk use derandomized $\mathsf{Send}/\mathsf{Receive}$ calls. In Encaps, the first Send call outputs a key k'; this is then hashed to derive (k, k_m), where k is the output key and k_m is a MAC key. Another Send call is then performed to bind the entire ciphertext to the URKE instance. Decaps follows analogously. Collision key ck is sampled at the beginning of each Encaps call to facilitate the security proof, namely so that $\mathsf{Receive}$ cannot be feasibly called before the corresponding Send call if randomness is not

```
Function Setup($1^\lambda$)
1 :  $\mathsf{pp}_{\Pi_u} \leftarrow \Pi_u.\mathsf{Setup}(1^\lambda)$
2 :  $\mathsf{pp} \leftarrow (\lambda, \mathsf{pp}_{\Pi_k})$
3 :  return pp

Function Gen($\mathsf{pp} = (\lambda, \mathsf{pp}_{\Pi_u})$)
1 :  $(\mathsf{st_A}, \mathsf{st_B}) \leftarrow \mathsf{Init}(\mathsf{pp}_{\Pi_u})$
2 :  $\mathsf{pk} \leftarrow \mathsf{st_A}$
3 :  $\mathsf{sk} \leftarrow (\mathsf{st_A}, \mathsf{st_B})$
4 :  return $(\mathsf{pk}, \mathsf{sk})$

Function Encaps(pk)
1 :  $\mathsf{ck} \leftarrow\$ \mathcal{K}$
2 :  $(\mathsf{pk}, k', \mathsf{ct}') \leftarrow \mathsf{Send}(\mathsf{pk}, (1, \mathsf{ck}))$
3 :  $(k, k_m) \leftarrow H(k')$
4 :  $t \leftarrow \mathsf{Tag}(k_m, (\mathsf{ck}, \mathsf{pk}, \mathsf{ct}'))$
5 :  $\mathsf{ct} \leftarrow (\mathsf{ck}, \mathsf{pk}, \mathsf{ct}', t)$
6 :  $(\mathsf{pk}, \cdot, \cdot) \leftarrow \mathsf{Send}(\mathsf{pk}, (2, \mathsf{ct}); 0)$
7 :  return $(\mathsf{pk}, k, \mathsf{ct})$

Function Decaps(sk, ct)
1 :  $(\mathsf{st_A}, \mathsf{st_B}) \leftarrow \mathsf{sk}$
2 :  $(\mathsf{ck}, \mathsf{pk}, \mathsf{ct}', t) \leftarrow \mathsf{ct}$
3 :  $(\mathsf{st_B}, k') \leftarrow \mathsf{Receive}(\mathsf{st_B}, (1, \mathsf{ck}), \mathsf{ct}')$
4 :  $(k, k_m) \leftarrow H(k')$
5 :  if $\neg \Pi_m.\mathsf{Ver}(k_m, (\mathsf{ck}, \mathsf{pk}, \mathsf{ct}'), t)$
6 :    return $(\mathsf{sk}, \perp)$
7 :  $(\mathsf{st_A}, \cdot, \mathsf{ct}'') \leftarrow \mathsf{Send}(\mathsf{st_A}, (2, \mathsf{ct}); 0)$
8 :  $(\mathsf{st_B}, \cdot) \leftarrow \mathsf{Receive}(\mathsf{st_B}, (2, \mathsf{ct}), \mathsf{ct}'')$
9 :  $\mathsf{sk} \leftarrow (\mathsf{st_A}, \mathsf{st_B})$
10 : return $(\mathsf{sk}, k)$

Function UpPk(pk, ad)
1 :  $(\mathsf{pk}, \cdot, \cdot) \leftarrow \mathsf{Send}(\mathsf{pk}, (0, \mathsf{ad}); 0)$
2 :  return pk

Function UpSk(sk, ad)
1 :  $(\mathsf{st_A}, \mathsf{st_B}) \leftarrow \mathsf{sk}$
2 :  $(\mathsf{st_A}, \cdot, \mathsf{ct}) \leftarrow \mathsf{Send}(\mathsf{st_A}, (0, \mathsf{ad}); 0)$
3 :  $(\mathsf{st_B}, \cdot) \leftarrow \mathsf{Receive}(\mathsf{st_B}, (0, \mathsf{ad}), \mathsf{ct})$
4 :  $\mathsf{sk} \leftarrow (\mathsf{st_A}, \mathsf{st_B})$
5 :  return sk
```

Fig. 11: LR-KUOWR-secure kuKEM Π_k (Definition 5) construction from LR-OW-secure URKE Π_u (Definition 9), LR-OT-SUF-secure MAC Π_m (Definition 3) and random oracle $H : \mathcal{K} \times \mathcal{K} \to \mathcal{K}$.

manipulated. Domain separation is employed to differentiate between different types Send/Receive calls in the associated data.

Theorem 4. *Let Π_m be a $(\ell+1, \ell, t', \epsilon_m)$-LR-OT-SUF secure MAC and Π_u a $(3q, 0, \ell_\mathsf{B}, 0, t', \epsilon_u)$-LR-OW secure URKE. Then, Π_k (Fig. 11) is $(q, \ell_\mathsf{B}, t, q^2/|\mathcal{K}| + q(\epsilon_m + \epsilon_u + \frac{1}{|\mathcal{K}|}))$-LR-KUOWR secure, where $t \approx t' \approx \tilde{t}$.*

Proof. The proof can be found in Appendix F.

Remark 6. Since a LR-OT-SUF MAC can be obtained from a random oracle as shown in Section 3.1, the above implies that LR-KUOWR kuKEM and LR-OW URKE are equivalent in the random oracle model.

7 Conclusion

This paper explores the security of unidirectional ratcheted key exchange in the bounded-leakage model under randomness manipulation. We formalize the notion of leakage-resilient key indistinguishability for URKE and provide a construction secure under this notion. Given the restrictions on the adversary that our notion of key indistinguishability imposes, we explore the weaker, but still useful, notion of leakage-resilient one-wayness for URKE. We then show that leakage-resilient one-way URKE and kuKEM are highly related, and consider how leakage-resilient kuKEM may be built. We conclude with some directions for future work.

– Extend the results to sesquidirectional and bidirectional ratcheting [PR18]. We hypothesise that a suitable notion of leakage resilience for key indistinguishability in these settings would enable the adversary to leak secret material on B (the receiver in URKE terminology). Unlike in URKE, B can heal from compromise (i.e., state exposure or leakage) in these primitives.
– Consider different notions of leakage and determine suitable performance/security trade-offs along with an appropriate leakage bound for practical use.
– Extend the results to the setting of group messaging: a natural direction would be to define and construct leakage-resilient continuous group key agreement (CGKA) [Alw+20a].

Acknowledgements. The research of Simone Colombo was partially supported by UKRI grant EP/X017524/1. Daniel Collins was supported in part by AnalytiXIN and by Sunday Group, Inc.

References

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. "The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol". In: *EUROCRYPT*. 2019.

[ADJ24] Martin R. Albrecht, Benjamin Dowling, and Daniel Jones. "Device-Oriented Group Messaging: A Formal Cryptographic Analysis of Matrix' Core". In: *SP*. 2024.

[ADW09a] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. "Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model". In: *CRYPTO*. 2009.

[ADW09b] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. "Survey: Leakage Resilience and the Bounded Retrieval Model". In: *ICITS*. 2009.

[Alw+20a] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. "Security Analysis and Improvements for the IETF MLS Standard for Group Messaging". In: *CRYPTO*. 2020.

[Alw+20b] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. "Continuous Group Key Agreement with Active Security". In: *TCC*. 2020.

[Bar+23] Khashayar Barooti, Daniel Collins, Simone Colombo, Loïs Huguenin-Dumittan, and Serge Vaudenay. "On Active Attack Detection in Messaging with Immediate Decryption". In: *CRYPTO*. 2023.

[BCG23] David Balbás, Daniel Collins, and Phillip Gajland. "WhatsUpp with Sender Keys? Analysis, Improvements and Security Proofs". In: *ASIACRYPT*. 2023.

[Bel+17] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. "Ratcheted Encryption and Key Exchange: The Security of Messaging". In: *CRYPTO*. 2017.

[BG21] Colin Boyd and Kai Gellert. "A Modern View on Forward Security". In: *Comput. J.* (2021).

[BOS17] Mihir Bellare, Adam O'Neill, and Igors Stepanovs. "Forward-Security Under Continual Leakage". In: *CANS*. 2017.

[BRV20a] Fatih Balli, Paul Rösler, and Serge Vaudenay. "Determining the Core Primitive for Optimally Secure Ratcheting". In: *ASIACRYPT*. 2020.

[BRV20b] Fatih Balli, Paul Rösler, and Serge Vaudenay. *Determining the Core Primitive for Optimally Secure Ratcheting*. Cryptology ePrint Archive, Paper 2020/148. 2020. URL: https://eprint.iacr.org/2020/148.

[CCG16] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. "On post-compromise security". In: *CSF*. 2016.

[Cha+23] Suvradip Chakraborty, Harish Karthikeyan, Adam O'Neill, and C. Pandu Rangan. "Forward Security Under Leakage Resilience, Revisited". In: *CANS*. 2023.

[CPR17] Suvradip Chakraborty, Goutam Paul, and C. Pandu Rangan. "Efficient Compilers for After-the-Fact Leakage: From CPA to CCA-2 Secure PKE to AKE". In: *ACISP*. 2017.

[CS98] Ronald Cramer and Victor Shoup. "A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack". In: *CRYPTO*. 1998.

[Dim+21] Antonio Dimeo, Felix Gohla, Daniel Goßen, and Niko Lockenvitz. "SoK: Multi-Device Secure Instant Messaging". In: *Cryptology ePrint Archive* (2021).

[Dod+10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana Lopez-Alt, and Daniel Wichs. *Efficient Public-Key Cryptography in the Presence of Key Leakage*. Cryptology ePrint Archive, Paper 2010/154. 2010. URL: https://eprint.iacr.org/2010/154.

[Dod+10b] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. "Cryptography against Continuous Memory Attacks". In: *FOCS*. 2010.

[Dod+10c] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. “Efficient Public-Key Cryptography in the Presence of Key Leakage”. In: *ASIACRYPT*. 2010.

[Dow+22] Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. “Strongly Anonymous Ratcheted Key Exchange”. In: *ASIACRYPT*. 2022.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. “Leakage-Resilient Cryptography”. In: *FOCS*. 2008.

[Dzi06] Stefan Dziembowski. “Intrusion-Resilience Via the Bounded-Storage Model”. In: *TCC*. 2006.

[EM19] Ksenia Ermoshina and Francesca Musiani. ““Standardising by running code”: the Signal protocol and de facto standardisation in end-to-end encrypted messaging”. In: *Internet Histories* (2019).

[Fan+10] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. “State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures”. In: *HOST*. 2010.

[Gen+16] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. “ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels”. In: *CCS*. 2016.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. In: *CHES*. Vol. 2162. Lecture Notes in Computer Science. Springer, 2001, pp. 251–261.

[Haz+16] Carmit Hazay, Adriana López-Alt, Hoeteck Wee, and Daniel Wichs. “Leakage-Resilient Cryptography from Minimal Assumptions”. In: *J. Cryptol.* (2016).

[HL11] Shai Halevi and Huijia Lin. “After-the-Fact Leakage in Public-Key Encryption”. In: *TCC*. 2011.

[JS18] Joseph Jaeger and Igors Stepanovs. “Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging”. In: *CRYPTO*. 2018.

[Koc+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. “Spectre Attacks: Exploiting Speculative Execution”. In: *SP*. 2019.

[KR19] Yael Tauman Kalai and Leonid Reyzin. “A survey of leakage-resilient cryptography”. In: *Providing Sound Foundations for Cryptography*. 2019.

[Lip+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. “Meltdown: Reading Kernel Memory from User Space”. In: *USENIX Security*. 2018.

[LRW11] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. "Achieving Leakage Resilience through Dual System Encryption". In: *TCC*. 2011.

[MR04] Silvio Micali and Leonid Reyzin. "Physically Observable Cryptography (Extended Abstract)". In: *TCC*. 2004.

[PM16] Trevor Perrin and Moxie Marlinspike. *The Double Ratchet Algorithm*. https://signal.org/docs/specifications/doubleratchet/. Accessed: 25-10-2022. 2016.

[PR18] Bertram Poettering and Paul Rösler. "Towards Bidirectional Ratcheted Key Exchange". In: *CRYPTO*. 2018.

[RSS23] Paul Rösler, Daniel Slamanig, and Christoph Striecks. "Unique-Path Identity Based Encryption with Applications to Strongly Secure Messaging". In: *EUROCRYPT*. 2023.

[Sco+22] John Scott-Railton, Elies Campo, Bill Marczak, Bahr Abdul Razzak, Siena Anstis, Gözde Böcü, Salvatore Solimano, and Ron Deibert. *CatalanGate: Extensive Mercenary Spyware Operation against Catalans Using Pegasus and Candiru*. https://citizenlab.ca/2022/04/catalangate-extensive-mercenary-spyware-operation-against-catalans-using-pegasus-candiru/. Accessed: 22-05-2022. 2022.

[Spr+18] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. "Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices". In: *IEEE Commun. Surv. Tutorials* (2018).

[Wan+22] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. "Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86". In: *USENIX Security*. 2022.

[Yan+19] Guomin Yang, Rongmao Chen, Yi Mu, Willy Susilo, Fuchun Guo, and Jie Li. "Strongly leakage resilient authenticated key exchange, revisited". In: *Designs, Codes and Cryptography* (2019).

[Yan+23] Hailun Yan, Serge Vaudenay, Daniel Collins, and Andrea Caforio. "Optimal Symmetric Ratcheting for Secure Communication". In: *The Computer Journal* (2023).

[YV20] Hailun Yan and Serge Vaudenay. "Symmetric Asynchronous Ratcheted Communication with Associated Data". In: *IWSEC*. 2020.

A Deferred preliminaries

A.1 Notation

We use maps, or associative arrays, which associate keys with values: $m[\cdot] \leftarrow x$ defines a new map with values initially set to x, and $m[k]$ returns the element indexed by key k. Let $[n] = \{1, \ldots, n\}$, i.e., the set of integers between 1 and n. For two values b and b', $1_{b=b'}$ is 1 if $b = b'$ and 0 otherwise. For every leakage function f we assume that $f(\perp) \rightarrow \perp$. Furthermore, some security games might abort. There are two kinds of abortion, denoted by $\mathbf{abort}_{\mathsf{IND}}$ and **abort**, both of which are defined in Appendix A.2.

We define relations prefix-or-equal $\preceq$ and strictly-prefix $\prec$ over two strings. For instance, for stirngs $a, b = a||x, c = a||y$, where $x \neq y$, we have that, $a \preceq b$, $a \preceq c$, $b \npreceq a$, $c \npreceq a$, which means that a is a prefix of both b and c, but neither b, c is a prefix of the other.

We denote with $\mathcal{R}$ the set of randomness used by all our functions. $\mathcal{R}$ does not contain $\perp$. We refer to randomness honestly sampled by an oracle with "fresh randomness", whereas we call "manipulated randomness" the randomness that the adversary controls and provides to the oracles. Assigning to a variable a a value output by a randomized algorithm $A()$ is denoted by $a \leftarrow\$ A()$. Sampling uniformly at random an element s from a set S is indicated with $s \leftarrow\$ S$. Some randomized functions can take a randomness parameter r which is used instead of freshly sampled randomness if it is not $\perp$. They are written $f(a, \ldots, b; r)$ where f is the function, $a, \ldots, b$ its normal parameters, and r the randomness parameter. If r is set to $\perp$, these functions produce a randomized output, while they are fully deterministic if r is set to any value in $\mathcal{R}$. Most keys (except asymmetric ones) are assumed to be elements of the general key space $\mathcal{K}$. The set $\mathcal{K}$ depends on the security parameter λ and is part of the public parameters of our schemes. We assume that $\mathcal{K} \subseteq \mathcal{R}$, i.e., $\mathcal{K}$ is a subset of $\mathcal{R}$.

A.2 Security games

In our security games, we use aborts and logs, which we detail below.

We can divide the security games that appear in this work in two sets: guessing and indistinguishability games. In a guessing game the adversary outputs a whole secret, e.g., a symmetric key, and we say that the scheme for which game is defined is secure if the success probability in guessing the secret is low for any efficient adversary. In indistinguishability games the adversary's goal is to guess a uniformly random bit based on its knowledge and the double of the distance to 1/2 of the success probability of all time-bounded adversaries measures the security of the construction.

Abortion. Following the difference in the definition of the adversary's advantage, we use two notions of abortion. For guessing games, abort indicates that the game returns 0, that is the adversary loses the game. For indistinguishability games $\mathbf{abort}_{\mathsf{IND}}$ indicates that the game returns a uniformly random bit. The goal of these definitions is to ensure that the adversary cannot increase its winning probability by intentionally aborting the game. This is in particular necessary if it can check whether it is winning, which is for example the case for some state-guessing attacks on LR-KIND and LR-OW. Therefore, all our probabilities over game runs also include the aborted game runs and use the appropriate abortion methods to ensure the adversary cannot gain anything by aborting.

Logs. Our games use several oracles, and some of these oracles log the inputs on which they are called and other information. We always assume that the values written in these logs are suitably encoded and cannot be confused with the other syntax used in these logs (for example delimiting characters or oracle names).

B On the use of an LR-OT-SUF-secure MAC scheme

The LR-KIND-secure URKE scheme that we present in Fig. 7 is essentially the same as in [BRV20a], but it uses an LR-OT-SUF-secure MAC instead of a classic OT-SUF-secure MAC. In this section we show that the construction from [BRV20a, Fig. 7] can be insecure under the partial leakage that we consider in our work.

Assume that the construction in Fig. 7 uses a classic OT-SUF-secure MAC scheme Π_m. Assume this MAC scheme has the property that leaking ℓ bits of the secret key allows for a full secret key recovery. This is a slightly stronger assumption than the MAC not being LR-OT-SUF, but does not contradict OT-SUF-security, and therefore allows to show that a OT-SUF-secure MAC as in the construction from [BRV20a, Fig. 7] is insufficient. Furthermore, we assume that within the same ℓ bits of leakage (or using prior exposure and leakage information) it is possible to get the kuKEM public key. This would not contradict the security of the construction, as we and [BRV20a] only use the security of the kuKEM which does not depend on the secrecy of its public key. The adversary proceeds as follows:

1. Leak Alice's state by querying LEAK-STATE$(f, \mathsf{A}, \perp)$ for the appropriate leakage function f and derive the entire MAC key (denoted as k_m in Fig. 7), which is possible by the assumption we made on the scheme.
2. Simulate the Send function until the call to the function H (Fig. 7, Send, lines 1 to 5 inclusive). The adversary can simulate because the kukem public key is assumed to be known and the previous step of the attack leaks the MAC key k_m, which is the only secret involved in lines 1 to 5.
3. Define a leakage function $f'_{k_e} \rightarrow H(k_e, k_s, \perp)$ by using the simulated k_e derived in the second step. Query LEAK-STATE$(f'_{k_e}, \mathsf{A}, \perp)$ to get the first bit b^* of k_o. This step is needed as k_s is secret and the adversary cannot simulate it.
4. Call RECEIVE$(\mathsf{ad}, \mathsf{ct})$ on the simulated ct and matching ad derived in the second step of the attack.
5. Call CHALL-KIND$(\mathsf{B}, (\mathsf{ad}, \mathsf{ct}))$ on the simulated ct and matching ad to obtain the challenge key k.

6. Output 0 if b^* equals the first bits of k and 1 otherwise. This enables the adversary to win the LR-KIND game of Fig. 6 with advantage 1/2.

This attack does not violate the sub-predicates that compose trivial-KIND. In particular, the adversary can mount the first step of the attack as it can leak up to leakage bound on Alice's state, whereas a full exposure of Alice's state is forbidden by predicate (P3): the adversary queries $\mathsf{RECEIVE}(\mathsf{ad}, \mathsf{ct})$ and $\mathsf{CHALL\text{-}KIND}(\mathsf{B}, (\mathsf{ad}, \mathsf{ct}))$ on the same transcript $\mathsf{tr}^* = (\mathsf{ad}, \mathsf{ct})$, where the variable tr^* is used in predicate (P3).

C Proof of Theorem 1

Proof. The proof follows broadly the strategy that Balli, Rösler and Vaudenay use to prove the security of their URKE construction [BRV20b, Appendix B], since their construction is the same as ours except that we do account for leakage.

A challenge key, by definition of our trivial attacks, is a key on which no bit was leaked and which was not revealed, so LEAK-KEY and EXP-KEY were never called on it. It cannot be leaked or exposed through the states of A or B via EXP-STATE or LEAK-STATE since in the construction of Fig. 6 it is never part of these states. The adversary $\mathcal{A}$ cannot gain any information directly on it. It has to guess whether the value returned by the CHALL-KIND oracle is the exchanged key or a random one, without any knowledge, which results in a success probability of exactly $\frac{1}{2}$, or call the random oracle H through which it is generated with the right inputs. Since the first case gives no advantage over random guessing, we focus on the second one and therefore assume in the following that the adversary makes a query to the random oracle H which results in the output of a valid challenge key.

G0.0: Let's do a first game hop from the base game **G0** depicted in Fig. 6 to a new game called **G0.0** to eliminate collisions in the random oracle. Game **G0.0** is identical to **G0**, except that the random oracle no longer randomly samples 4 keys from the set $\mathcal{K}$ when it is called on unseen input. Instead, it samples uniformly random keys in the set of all elements of $\mathcal{K}$ which were not already sampled by the oracle. Since the probability of a collision in any kind of keys output by the random oracle is upper bounded by $\frac{(4(q_h+q_s))^2}{|\mathcal{K}|}$ where q_h is the number of random oracle calls and q_s the number of SEND and RECEIVE calls the LR-KIND adversary makes (including those in leakage functions) during the game, the success probability of any adversary in distinguishing the games **G0** and **G0.0** is upper bounded by $\frac{(4(q_h+q_s))^2}{|\mathcal{K}|}$ also.

In **G0.0**, if the oracle makes a call to H which outputs a valid challenge key, it makes a call with exactly the inputs with which this challenge key was obtained during sending or receiving, since SEND and RECEIVE are the only oracles which access the exchanged key and they obtain it via a random oracle call. If this random oracle call was made during sending, the cases of sending without or with manipulated randomness are distinguished. If both occurred, only the call without randomness manipulation in the SEND call is considered.

Since it is possible that the random oracle was never queried on these inputs by SEND, but only by RECEIVE, a third case needs to be added. RECEIVE does not use randomness, therefore further distinctions are not needed. As no other oracle uses random oracle calls nor accesses the challenge key, the list of cases below is exhaustive, so that we can treat them separately and then sum the probabilities of the adversary to reach these cases to obtain an upper bound on $\epsilon_{\mathsf{LR\text{-}KIND}}$.

1. A call to SEND without randomness manipulation made the random oracle call.
2. A call to SEND with randomness manipulation made the random oracle call and no call to it without randomness manipulation did so.
3. The random oracle query was made during a call to RECEIVE and in no call to the SEND oracle.

In the remainder of the proof, we will show that Case 1. reduces to the KUOWR security of kuKEM, Case 2. to guessing a hidden key, and Case 3. to the LR-OT-SUF security of the MAC scheme. We first explain two steps in the reduction which are necessary for all three cases.

In the game **G0.0** (as in the base game **G0** from Fig. 6), the reduction runs the URKE construction to provide the oracles towards the adversary, and does not respond at all to its challenger. In the following step, we split the random oracle interface into two consistent random oracles H and G as in [BRV20b]. The adversary and the two leakage oracles LEAK-KEY and LEAK-STATE still interact with an oracle H to which they input k_e, k_s, tr triplets. All other oracles (but not the adversary) use the random oracle G which only takes as input a trace tr. Both oracles are maintained consistently: For each trace tr where keys k_e and k_s exist, G outputs the same value on tr as H outputs on the triplet of tr and its k_e and k_s values. Since G is used instead of H by the oracles, it is only called when keys k_e and k_s exist, which guarantees that this change does not modify the outputs of the random oracles. If H is called on a new triplet k_e, k_s, tr for which k_e and k_s do not match the oracle's keys for tr or the oracle does not yet know keys for that trace, H outputs a new random value and stores the triplet in order to give consistent answers for it. If G is called for the first time on a trace, then it is called on by an oracle which has defined k_s and k_e. If H was already queried for this trace with the same k_s and k_e, G gives consistent output, otherwise it samples a new one and stores it with the trace for future calls. The appendix of [BRV20b] has a schematic representation of this oracle splitting step.

G0.1: Similarly to [BRV20b], by keeping lists of keys of traces for which the EXP-KEY, EXP-STATE, LEAK-STATE or LEAK-KEY oracle or SEND with manipulated randomness or RECEIVE after leak or exposure on the receiver was called and which are therefore not completely unknown to the adversary, all other keys which were only used in calls to H which were replaced by calls to G can just never be sampled, since the adversary never learns anything on them. Furthermore, keys which the adversary learns through an oracle call can be sampled by the challenger only at the moment where it learns them, and then stored for potential H calls containing them. Therefore, the challenger only needs to know and fix k_e and k_s keys if one of these oracles is called for their trace. The game with these modified random oracles is denoted **G0.1**. Since **G0.1** is a rewriting of **G0.0** the adversary has the same success probability as in **G0.0**.

The remainder of the proof will treat the three cases in order, one after the other. In each of them, we will start from **G0.1**.

Case 1. Assume that the adversary makes at some moment of the game a call to the random oracle H with input k_e, k_s, tr such that SEND also made such a call with the same inputs when treating a call where randomness was not manipulated.

G1.1: The game creates URKE sender and receiver states as described in the construction, and uses it to simulate all URKE oracles to the adversary. The random oracle is split as described above. In **G1.1** the adversary has the same success probability as in **G0.1** assuming Case 1.

G1.2: We define a reduction for KUOWR a adversary that simulates as follows. It replaces the public kuKEM key in the sender state by the public kuKEM key of its KUOWR challenger, and removes the private kuKEM key from the receiver state. The oracles are simulated as follows:

– Calls to LEAK-STATE and EXP-STATE on the sender are unchanged.
– RECEIVE is simulated by using the DEC oracle of the KUOWR challenger to get the encapsulated key, and using the update key and the UPSK oracle of the KUOWR challenger to update the kuKEM secret key so that it stays synchronized with the public key which encapsulated the received key.
– EXP-STATE on the receiver requires to use the expose oracle of the KUOWR challenger.
– LEAK-STATE on the receiver is simulated by exposing the secret key and then leaking the required bit of it. This makes sense since leaking on the receiver is only authorized after the generation of all challenge keys.
– SEND is simulated by using the public kuKEM key for encapsulating a random key, and then updating the public key accordingly via the UPPK oracle. Therefore, all encapsulated keys are known at their generation, the array $\mathsf{key}[\cdot,\cdot]$ can be maintained and the CHALL-KIND, LEAK-KEY and EXP-KEY oracles are unchanged.

Since the key pair is subject to exactly the same operations at the same moments in the game execution (only that these operations now mostly take place in oracles provided by the challenger), this game change does not impact the adversary, which therefore still has the same success probability as in game **G1.1** assuming Case 1.

G1.3: We define a similar reduction as above except that SEND is simulated differently: Now the ENC oracle of the KUOWR game is used for encapsulation. Furthermore, SEND calls with manipulated randomness can be simulated via ENC calls with manipulated randomness. This means that the encapsulated keys are not known to the reduction. However, the $\mathsf{key}[\cdot,\cdot]$ array can still be maintained, since its contents are keys k_o output by G which does not require the exchanged key as input. Thus the simulation of CHALL-KIND, LEAK-KEY and EXP-KEY is straightforward. The only two cases in which the adversary can access an encapsulated key k_e is by leaking on the sender state while simulating encapsulation with manipulated randomness, or if it makes a SEND query with manipulated randomness. The first case is trivial since the adversary provides the function to be evaluated which computes itsel the encapsulated key, and in the second case, the DEC query in the KUOWR game can be used to give the adversary the necessary knowledge, since a key generated with manipulated randomness is not a KUOWR challenge key. Therefore the adversary still has the same success probability as in game **G1.2** assuming Case 1.

When the adversary now makes a random oracle query to H for a trace tr which corresponds to a challenge key which was encapsulated using fresh randomness, this is a key which is also a challengeable kuKEM key, since there was no decapsulation query on it, and the receiver B was not leaked nor exposed before its generation. Therefore, the reduction can take the k_e value in the call to H, and submit it to its KUOWR challenger in order to win the KUOWR game. As the LR-KIND adversary runs in time t, the reduction is ran by a KUOWR adversary running in time $t' \approx t$ since it does no expensive operation except the leakage function simulation and running the adversary. It makes at most $2q+1$ queries to KUOWR oracles (exactly one SOLVE and zero LEAKSK calls, 2 calls per SEND and two per call to RECEIVE, at most one to EXP per leakage query). It solves KUOWR as often as the LR-KIND adversary solves LR-KIND via Case 1. Therefore, the probability of the LR-KIND adversary to call H so that it outputs a valid challenge key (this is, to reach Case 1.) is bounded by $\epsilon_{\mathsf{KUOWR}}$, so its change to win the LR-KIND game by Case 1. is upper bounded by $\epsilon_{\mathsf{KUOWR}}$.

Case 2. Assume that the adversary makes at some moment of the game a call to the random oracle H with input k_e, k_s, tr such that the SEND oracle also made such a call with the same inputs when treating a call where randomness was manipulated, and this call yields a challenge key. Let the sender state associated to tr where this random oracle call is made be T. Since the call yields a challenge key, there is a state S of the sender corresponding to trace tr' such that:

– From S until T randomness is always manipulated:

$$\nexists\, i, \mathsf{tr}^* \colon (\mathsf{tr}' \prec \mathsf{tr}^* \preceq \mathsf{tr}) \wedge (\mathsf{log}[i] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}^*, \cdot, \cdot, \bot)).$$

– No state between S and T (both included) is exposed via EXP-STATE on the sender: $\nexists\, i, \mathsf{tr}^* \colon (\mathsf{tr}' \prec \mathsf{tr}^* \preceq \mathsf{tr}) \wedge (\mathsf{log}[i] = (\text{“}\mathsf{stexp}\text{”}, \mathsf{A}, \mathsf{tr}^*))$.
– S was generated with fresh randomness: $\exists\, i \colon \mathsf{log}[i] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}', \cdot, \cdot, \bot)$.
– No LEAK-STATE on the sender state occurs between S and T:

$$|\{i \colon (\exists\, \mathsf{tr}^* \colon \mathsf{log}[i] = (\text{“}\mathsf{stleak}\text{”}, \mathsf{A}, \cdot, \mathsf{tr}^*))\}| = 0.$$

Since there is no leakage involved between S and T, the argumentation that k_s^T is unknown to the adversary is the same as in [BRV20b].

__G2.1__: The challenger wants the adversary to guess a random value K. The game completely simulates an instance of URKE to the adversary. The random oracle is split as above. This game is identical to **G0.1** in Case 2., so the adversary has the same success probability as in Case 2.

__G2.2__: The challenger guesses for which state S the adversary will make the H call, and substitutes K for k_s^S. This is possible since it only uses G and not H on S since no leakage nor exposure on this state is possible (since a send with manipulated randomness on it produces a valid challenge key), and that it was obtained through a random oracle query to G. If the guess is correct, **G2.2** is just a rewriting of **G2.1**, so the adversary has the same success probability in both. The probability that the guess is correct is $\frac{1}{q}$, where q is the number of H queries the adversary makes.

To see that S was generated through a G query, note that the last state before S which was generated using fresh randomness had his generation therefore simulated by G (as the key k was not sampled since not required). Since itself and all following k_s were never exposed nor leaked, they were never generated, but their creation was postponed to the first oracle query requiring it (leak or expose query) which never occurred. Therefore, the challenger does not need to know k_s^S to correctly simulate the URKE construction to the adversary. The H query of the adversary then gives the random value it guessed to the challenger, which therefore wins as often as it guessed S correctly among a polynomial number of possible states (as many as send calls) and the adversary wins. This gives a success probability of $\frac{q}{|\mathcal{K}|}$, independently of the adversary's runtime.

Case 3. Assume that the adversary will make at some moment of the game a call to the random oracle H with input k_e, k_s, tr such that the receive oracle also made such a call with the same inputs and this call yields a challenge key, but no send call ever used them. In this case, a message was accepted which was never sent. We reduce this to the strong unforgeability of MAC under bounded leakage of bound ℓ (which is equal to the leakage bound of the URKE construction).

__G3.1__: The game runs the URKE construction and all oracles for the adversary. The random oracle is split as above. This game is identical to **G0.1** in Case 3., so the adversary has the same success probability in Case 3.

__G3.2__: We define an LR-OT-SUF adversary that simulates as follows. The adversary guesses which of the $\leq q$ receivers state S containing the MAC key k_m^S which will be used for receiving a message never sent, if any. Do not sample k_m^S, but use the Tag and Ver oracles of the LR-OT-SUF challenger instead. If only RECEIVE is called on it, there is no need to sample it, since G is used. The state of the receiver B containing it cannot be exposed nor leaked since it is B's state before receiving a challenge key. However, if there is a state of the sender containing it, this state could have been leaked. If there is leakage on this state, sample all other components of the sender's state and call the LEAK-KEY oracle of the LR-OT-SUF challenger. The no-impersonation rule forbids the exposure of this state, and the definition of P6 prevents more than ℓ bits of leakage in the case of impersonation. As a consequence, in the case where the state on which the winning H call occurs is guessed correctly, the adversary's success probability in this game is the same as in game **G3.1**. The guess is correct with probability $\frac{1}{q_s}$, where q_s is the number of SEND and RECEIVE calls the adversary makes. Therefore, the probability that the guess was correct and the adversary succeeds is upper bounded by $\frac{1}{q_s} \cdot \epsilon_{\mathsf{LR\text{-}KIND}}$.

When the message not created by the SEND oracle is received by RECEIVE, it is not identical to a previous TAG query (since only SEND is simulated by TAG queries) and therefore permits to win the LR-OT-SUF game. This means that $q_s \Pr[\mathsf{LR\text{-}OT\text{-}SUF}^{\ell}_{\mathsf{reduction}}(1^{\lambda}) \rightarrow 1] \geq \Pr[(\mathsf{LR\text{-}KIND}^{\ell}_{\mathcal{A}}(1^{\lambda}) \rightarrow 1) \wedge (\text{Case 3.})]$. The reduction respects the leakage bound ℓ_{A}, makes only q queries to the LR-OT-SUF oracles (one per leak and one per SEND or RECEIVE using the MAC key from the challenger), and runs in time $\tilde{t} \approx t$ where t is the runtime of the adversary, since the only potentially costly operations it runs are the execution of the adversary and some leakage function evaluations. Its success probability is therefore bounded by $\epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}$. As a consequence, the probability that the adversary wins LR-KIND through Case 3. is bounded by $q_s \cdot \epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}$. □

D Proof of Theorem 2

Proof. In this proof, the number of random oracle queries by the LR-OW adversary is denoted by q_h and the number of SEND/RECEIVE queries by q_s.

The main difference between LR-OW and LR-KIND is that an LR-OW adversary can win by correctly guessing the challenge key. There are therefore two ways in which the adversary can win the game: either it makes a random oracle query outputting it (Case -1.), or it does not make such a query and guesses the key with the available knowledge (Case 0.).

In Case 0., its success probability is $\frac{2^{\ell_{\mathsf{A}}+\ell_{\mathsf{B}}+\ell_{\mathsf{k}}}}{|\mathcal{K}|}$, since by construction of the URKE scheme the key is sampled uniformly at random in $\mathcal{K}$ and the leakage bounds are enforced by the trivial-OW function. This case simply does not exist in the LR-KIND game, since the adversary there could not check such a guess if it made one, and therefore guessing a challenge key correctly does not affect its chance of succeeding.

The remainder of the proof will bound the success probability in Case -1. which is similar to the LR-KIND proof.

If it does not simply guess the challenge key as in Case 0., the adversary only has the possibility to obtain the challenge key either by making a query to the random oracle with the same inputs as the SEND or RECEIVE oracles use to obtain it, or by exploiting a collision in the random oracle.

We can use the same argument as in the LR-KIND proof to move to a game **G0.0** without collisions, so the success probability of an adversary distinguishing between the original game (in Case -1.) and **G0.0** is bounded by $\frac{(4(q_h+q_s))^2}{|\mathcal{K}|}$. This game hop therefore adds a term $\frac{(4(q_h+q_s))^2}{|\mathcal{K}|}$ to our final bound on the success probability of the adversary.

Furthermore, by the same reasoning as the LR-KIND proof uses for game **G0.0** (that means, assuming no collisions and no random guessing of the key), there are three cases in which the adversary makes a call to H which outputs a challenge key, which was therefore also output in a H call in a SEND or RECEIVE call:

1. A call to SEND without randomness manipulation made the random oracle call.
2. A call to SEND with randomness manipulation made the random oracle call and no call to it without did so.
3. The random oracle query was made during a call to RECEIVE and in no call to the SEND oracle.

In addition, we can, as in the LR-KIND proof, split the random oracle in **G0.0** into two oracles H and G with consistent outputs such that G is used by SEND and RECEIVE and only requires a trace as input, and H is used by the leak oracles and the adversary and requires a triplet of the two correct keys k_e and k_s for a given trace tr in addition to that trace. This does not impact the adversary's success probability. Finally, it is possible to use lazy sampling for all keys k_s and k_e, which are therefore only generated if any computation different from a random oracle call is done on them. This again does not impact the success probability. The resulting came is denoted **G0.1** and serves as a base for the three separate reductions we will construct to capture the three cases listed above.

Case 1. **G1.1**: The challenger creates URKE sender and receiver states as described in the construction, and uses it to simulate all URKE oracles to the adversary. The random oracle is split as described above. In **G1.1** the adversary has the same success probability as in **G0.1** assuming Case 1..

G1.2: We define an adversary that simulates as follows. The adversary replaces the public kuKEM key in the sender state by the public kuKEM key of its LR-KUOWR challenger, and removes the private kuKEM key from the receiver's state. The oracles are simulated as in game **G1.2** of the LR-KIND proof, except for the simulation of LEAK-STATE on B, which is done by using the LEAKSK oracle in the LR-KUOWR game. To do so, the other keys contained in B's state must be generated and incorporated into the leakage function given by the adversary, to obtain the corresponding leakage function for the LEAKSK oracle. This game change does not change the adversary's success probability, which is the same as in game **G1.1** assuming Case 1.

G1.3: In this hop, SEND and LEAK-STATE on B are simulated differently. SEND is simulated as in game **G1.3** of the LR-KIND proof. For LEAK-STATE on party B, up to ℓ_{B} bits of leakage are simulated by using the LEAKSK oracle in the LR-KUOWR game. In order to do so, the other keys contained in B's state must be generated and incorporated into the leakage function given by the adversary, in order to obtain the corresponding leakage function for the LEAKSK oracle. If the adversary requires more than ℓ_{B} bits of leakage on a trace, the kuKEM secret key is exposed and used to compute the additional leakage. As in the

$\mathsf{LR\text{-}KIND}$ proof, the $\mathsf{key}[\cdot,\cdot]$ array can be maintained. Differently to $\mathsf{LR\text{-}KIND}$ the adversary can now make $\mathsf{LEAK\text{-}STATE}$ queries on the state of A before a challenge key is sent. Since challenge keys in Case 1. are always generated with fresh randomness, this is not relevant here. Thus the adversary still has the same success probability as in game **G1.2** assuming Case 1.

As in the $\mathsf{LR\text{-}KIND}$ proof, observing a query to H resulting in a valid challenge key generated by sending without randomness manipulation allows the adversary to win the $\mathsf{LR\text{-}KUOWR}$ game. Therefore, it always wins the KUOWR game if the adversary wins the $\mathsf{LR\text{-}OW}$ game via Case 1. The reduction is a $\mathsf{LR\text{-}KUOWR}$ adversary with respect to the kuKEM which runs in time $t' \approx t$ if the adversary runs in time t, since it performs no costly operations besides running the adversary and executing some of the leakage functions, which are both counted in t. It makes at most two $\mathsf{LR\text{-}KUOWR}$ oracle queries per $\mathsf{LEAK\text{-}STATE}$, SEND or $\mathsf{RECEIVE}$ call of the $\mathsf{LR\text{-}OW}$ adversary. Therefore, if the $\mathsf{LR\text{-}OW}$ adversary calls q $\mathsf{LR\text{-}OW}$ oracles in total, the adversary makes at most $2q + 1$ $\mathsf{LR\text{-}KUOWR}$ oracle queries (including one to SOLVE) and runs in time t' while respecting leakage bound ℓ_{A}. As a consequence, its success probability is bounded by $\epsilon_{\mathsf{LR\text{-}KUOWR}}$. This bounds the success probability of the adversary in Case 1. to at most $\epsilon_{\mathsf{LR\text{-}KUOWR}}$.

<u>Case 2.</u> In Case 2., we assume that the adversary will make at some moment of the game a call to the random oracle H with input k_e, k_s, tr such that the SEND oracle also made such a call with the same inputs when treating a call where randomness was manipulated (and never with fresh randomness), and this call yields a challenge key. Let the sender state associated to tr where this random oracle call is made be T. Since the call yields a challenge key, there is a state S of the sender corresponding to trace tr' such that:

- From S until T, randomness is always manipulated:
  $\nexists\, i, \mathsf{tr}^* \colon (\mathsf{tr}' \prec \mathsf{tr}^* \preceq \mathsf{tr}) \wedge (\mathsf{log}[i] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}^*, \cdot, \cdot, \bot)$
- No state between S and T (both included) is exposed via $\mathsf{EXP\text{-}STATE}$ on the sender:
  $\nexists\, i, \mathsf{tr}^* \colon (\mathsf{tr}' \prec \mathsf{tr}^* \preceq \mathsf{tr}) \wedge (\mathsf{log}[i] = (\text{“}\mathsf{stexp}\text{”}, \mathsf{A}, \mathsf{tr}^*)$
- S was generated with fresh randomness:
  $\exists\, i \colon \mathsf{log}[i] = (\text{“}\mathsf{send}\text{”}, \mathsf{A}, \mathsf{tr}', \cdot, \cdot, \bot)$
- $\mathsf{LEAK\text{-}STATE}$ on the sender state occurs at most ℓ_{A} times between S and T:
  $|\{i \colon (\exists\, \mathsf{tr}^* \colon \mathsf{log}[i] = (\text{“}\mathsf{stleak}\text{”}, \mathsf{A}, \cdot, \mathsf{tr}^*))\}| \leq \ell_{\mathsf{A}}$

Since in the $\mathsf{LR\text{-}OW}$ game there is leakage between S and T, we cannot argue that k_s^T is unknown to the adversary as in the $\mathsf{LR\text{-}KIND}$ proof or [BRV20b, Appendix B]. Therefore, we cannot reduce to guessing a completely unknown state key, but only to guessing a state key on which $\ell_{\mathsf{A}} + \ell_{\mathsf{B}}$ bits of leakage are allowed.

<u>**G2.1**</u>: The challenger wants the adversary to guess a random value K. For this, it samples K from $\mathcal{K}$. Then, it allows the adversary to access at least $\ell_{\mathsf{A}} + \ell_{\mathsf{B}}$ times a $\mathsf{LEAK\text{-}K}$ oracle which checks and increments a counter counting the calls to it (and returns $\bot$ if the check failed) and returns one bit equal to $f(K)$

otherwise, where f is a function provided by the adversary to the oracle which always outputs a bit. The game completely simulates an instance of URKE to the adversary. The random oracle is split as above.

From the adversary's perspective, this game is identical to **G0.1** in Case 2., so the adversary has the same success probability.

G2.2: The game guesses for which state S the adversary will make the H call, and substitutes K for k_s^S. This is possible since both of these are uniformly randomly sampled from $\mathcal{K}$, either by a random oracle or by the challenger. The only oracles using state keys is H, and it is only called on k_s by the adversary (which does not need to be simulated) or a LEAK-STATE call. LEAK-STATE can be called at most $\ell_\mathsf{A} + \ell_\mathsf{B}$ times on a state k_s prior to a challenge key: By ℓ_A calls to LEAK-STATE on A and ℓ_B on B. The LEAK-STATE oracle can be simulated in these cases by calling the LEAK-K oracle of the challenger with a leakage function where all other state components of the leaked party are inserted where they are used, and the random oracle (including all its state) is incorporated in the leakage function in case it is used by it.

If the challenger guesses correctly, **G2.2** is just a rewriting of **G2.1**, so the adversary has the same view and same success probability in both cases. Furthermore, in all cases where the adversary succeeds in the LR-OW game via Case 2. and the challenger guessed correctly, game **G2.2** allows an adversary to win the key-guessing with leakage game with probability of at most $\frac{2^{\ell_\mathsf{A}+\ell_\mathsf{B}}}{|\mathcal{K}|}$. Since the challenger guesses correctly with probability $\frac{1}{q_h}$ where q_h is the number of H queries the adversary makes (including those within leakage functions), this bounds the success probability of the adversary in Case 2. to $q_h \cdot \frac{2^{\ell_\mathsf{A}+\ell_\mathsf{B}}}{|\mathcal{K}|}$.

Case 3. As for LR-KIND, we reduce the case where some message is received and accepted and decapsulated into a challenge key which has never been sent, to the LR-OT-SUF security of the MAC scheme for bound.

G3.1: This is the same as in the LR-KIND proof and also does not impact the adversary at all.

G3.2: This is is also very similar to the LR-KIND case. The main difference is that LEAK-KEY and LEAK-STATE on B need to be simulated in the case where the reduction guessed correctly.

LEAK-KEY leaks information on the key output by the random oracle and not on the exchanged key, and it can therefore be simulated by uniformly randomly sampling a challenge key and leaking on it.

LEAK-STATE on B is simulated by querying the LEAK-KEY oracle of the LR-OT-SUF game with the leakage function received for leakage on B in which all keys except the MAC key are replaced by their actual values (which the reduction always knows).

Therefore, this game change is only a rewriting in the case where the reduction guessed correctly, which happens with probability $\frac{1}{q_s}$.

As a consequence, in the case where the state on which the winning H call occurs is guessed correctly, the adversary's success probability in this game is the same as in game **G3.1**. The guess is correct with probability $\frac{1}{q_s}$.

In game **G3.2**, if the reduction guessed correctly for which MAC key among the at most q_s existing ones the adversary will make the H query which allows it to obtain a challenge key, the reduction wins the LR-OT-SUF game by passing the message on which RECEIVE produced this challenge key to the LR-OT-SUF challenger. The reduction runs in time $\tilde{t} \approx t$ since it does not do any costly operations except simulating and leakage function executions, which both are counted in t. Furthermore, it makes one LR-OT-SUF oracle query per RECEIVE and LEAK-STATE on B call, so in total at most q LR-OT-SUF oracle queries. It therefore is an LR-OT-SUF adversary running in time $\tilde{t}$ making at most q oracle queries and respecting leakage bound $\ell_{\mathsf{A}} + \ell_{\mathsf{B}}$, and as a consequence its success probability is at most $\epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}$. Thus, the adversary has a success probability of at most $q_s\epsilon_{\mathsf{LR\text{-}OT\text{-}SUF}}$ in Case 3. □

E Deferred material for LR-KUOWR-secure kuKEM and proofs for Section 6.1

E.1 HIBE correctness definition

We provide a formal correctness definition for hierarchical identity-based encryption (HIBE), captured in Fig. 12.

```
Game $\mathsf{CORRECT}_{\mathcal{A}}(1^\lambda)$
1: $(\mathsf{pp}, \mathsf{mk}) \leftarrow \mathsf{Setup}(1^\lambda)$
2: $i, j \leftarrow 0$
3: $\mathsf{E}[\cdot], \mathsf{sks}[\cdot] \leftarrow \perp$
4: $\mathsf{win} \leftarrow 0$
5: $\mathcal{A}^{\mathcal{O}}(\mathsf{st})$
6: return $\mathsf{win}$

$\mathsf{GEN}(\vec{id})$
1: $i \leftarrow i + 1$
2: $\mathsf{sks}[i] \leftarrow (\mathsf{Gen}(\vec{id}, \mathsf{mk}), \vec{id})$
3: return $\mathsf{sks}[i]$

$\mathsf{ENC}(m, \vec{id})$
1: $j \leftarrow j + 1$
2: $\mathsf{E}[j] \leftarrow (\mathsf{Enc}(m, \vec{id}), m, \vec{id})$
3: return $\mathsf{E}[j]$

$\mathsf{DEL}(id, i')$
1: require $i' \in [1, i]$
2: $(\mathsf{sk}, \vec{id}) \leftarrow \mathsf{sks}[i]$
3: $\mathsf{sk}' \leftarrow \mathsf{Del}(\vec{id}, \mathsf{sk}, id)$
4: $\mathsf{sks}[i] \leftarrow (\mathsf{sk}', \vec{id} \mid\mid id)$
5: return $\mathsf{ct}$

$\mathsf{DEC}(i', j')$
1: $(\mathsf{ct}, m, \vec{id}_j) \leftarrow \mathsf{E}[j']$
2: $(\mathsf{sk}, \vec{id}_i) \leftarrow \mathsf{sks}[i']$
3: require $\vec{id}_i \preceq \vec{id}_j$
4: if $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) \neq m$
5:     $\mathsf{win} \leftarrow 1$
6: return
```

Fig. 12: HIBE correctness game.

Definition 13 (HIBE correctness). *Consider the* $\mathsf{CORRECT}$ *game of Fig. 12. An HIBE scheme is* correct *if for all (possibly unbounded) adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr[\mathsf{CORRECT}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] = 0.$$

E.2 True-simulation extractable NIZKs

We follow the work of Dodis, Haralambiev, Lopez-Alt and Wichs [Dod+10c] in defining the notions below.

Definition 14. *Let R be an NP relation on pairs (x, y) corresponding to the language $L_R = \{y : \exists x : (x, y) \in R\}$. A* non-interactive zero-knowledge (NIZK) argument *for a relation R comprises the following efficient algorithms:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{tk})$ *takes a unary string 1^λ and outputs a common reference string* crs *and a trapdoor key* tk*.*
- $\mathsf{Prove}(\mathsf{crs}, x, y) \to \pi$ *takes a common reference string* crs *and a pair $(x, y) \in R$ and outputs a proof π.*
- $\mathsf{Ver}(\mathsf{crs}, y, \pi) \to \mathsf{acc}$ *takes a common reference string* crs*, a value y and a proof π and outputs an acceptance bit* $\mathsf{acc} \in \{\mathsf{true}, \mathsf{false}\}$*.*

Sometimes we omit crs *as input from* Prove *and* Ver *when it is clear from context.*

Definition 15 (Completeness). *We say that NIZK argument Π for NP relation R is* complete *if for any $(x, y) \in R$, if $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\pi \leftarrow \mathsf{Prove}(x, y)$, then $\mathsf{Ver}(y, \pi) = 1$.*

Definition 16 (Composable Zero-Knowledge). *A NIZK is (t, ϵ)-*CZK*-secure for security parameter λ and NP relation R on pairs (x, y) corresponding to the language $L_R = \{y \colon \exists x \colon (x, y) \in R\}$ if there exist an efficient simulator* Sim *such that for all adversaries $\mathcal{A}$ which run in time at most t we have*

$$2 \cdot \left|\Pr[\mathsf{CZK}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] - 1/2\right| \leq \epsilon,$$

where the probability is taken over all the random coins that the challenger and the adversary use, and the game CZK *is defined in Fig. 13.*

Definition 17 (True-Simulation f-Extractability [Dod+10c]). *A NIZK is (t, q, ϵ)-f-*TSE*-secure for security parameter λ, NP relation R on pairs (x, y) corresponding to the language $L_R = \{y \colon \exists x \colon (x, y) \in R\}$ and function f if there exist algorithms $(\mathsf{crs}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{Setup}^*(1^\lambda)$ and $z \leftarrow \mathsf{Ext}(y, \phi, \mathsf{ek})$ such that for all adversaries $\mathcal{A}$ which run in time at most t and make at most q queries to* SIM *we have*

$$\Pr[f\text{-}\mathsf{TSE}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] \leq \epsilon,$$

where the probability is taken over all the random coins that the challenger and the adversary use, and the game f-TSE *is defined in Fig. 14.*

```
Game $\mathsf{CZK}_{\mathcal{A}}(1^\lambda)$
1 :  $b \leftarrow\$ \{0,1\}$
2 :  $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 :  $(x, y, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}, \mathsf{tk})$
4 :  $\pi_0 \leftarrow \mathsf{Prove}(x, y);\ \ \pi_1 \leftarrow \mathsf{Sim}(y, \mathsf{tk})$
5 :  $b' \leftarrow \mathcal{A}(\pi_b, \mathsf{st})$
6 :  return $1_{b=b'}$
```

Fig. 13: CZK game that we use in Definition 16.

```
Game $f\text{-}\mathsf{TSE}_{\mathcal{A}}(1^\lambda)$
1 :  $(\mathsf{crs}, \mathsf{tk}, \mathsf{ek}) \leftarrow \mathsf{Setup}^*(1^\lambda)$
2 :  $(y^*, \phi^*) \leftarrow \mathcal{A}^{\mathsf{SIM}}(1^\lambda)$
3 :  $z^* \leftarrow \mathsf{Ext}(y^*, \phi^*, \mathsf{ek})$
4 :  if $y^*$ previously input to SIM by $\mathcal{A}$ then return 0
5 :  if $\mathsf{Ver}(y^*, \phi^*) = 0$ then return 0
6 :  if $\forall x' : f(x') = z^*, R(x', y)' = 0$ then return 1
```

Fig. 14: f-TSE game that we use in Definition 17, where SIM is a simulation oracle that outputs simulated proofs. For simplicity that we omit labels, considered by [Dod+10c], from our notion.

E.3 Proof for Theorem 3

Proof. Our proof strategy is very similar to that of Dodis, Haralambiev, Lopez-Alt and Wichs to prove security of their LR-IND-CCA-secure public-key encryption scheme in [Dod+10a, Appendix A.3]. We proceed by a hybrid argument and construct games **G0** to **G2**; Let $\Pr[S_i]$ be the probability that **Gi** outputs 1 for $i \in [0, 2]$.

G0: This is the LR-OW-CCA game instantiated with Π^*. We have:

$$\Pr[S_0] = \Pr[\mathsf{LR\text{-}OW\text{-}CCA}^{\ell}_{\mathcal{A}}(1^\lambda) \Rightarrow 1]$$

G1: This is the same as **G0**, except the call "$\mathsf{ct}^* \leftarrow \mathsf{Enc}(m^*, \vec{id})$" in the CHALL-OW oracle is modified to replace the call "$\pi \leftarrow \Pi_n.\mathsf{Prove}(\mathsf{pp.crs}, ((m, r), (\vec{id}, \mathsf{ct}))$" by the output of algorithm $\mathsf{Sim}((\vec{id}, \mathsf{ct}), \mathsf{tk})$, where Sim is defined in the CZK game. Since Π_n is CZK-secure it follows that

$$|\Pr[S_0] - \Pr[S_1]| \leq \epsilon_z$$

G2: This is the same as **G1**, except for the following changes:

– The call "$(\mathsf{pp}, \mathsf{mk}) \leftarrow \Pi_h.\mathsf{Setup}(1^\lambda)$" is replaced by a call "$(\mathsf{pp}, \mathsf{mk}, \mathsf{ek}) \leftarrow \Pi_h.\mathsf{Setup}^*(1^\lambda)$" where Setup^* is defined in the f-TSE game.
– On decryption query $\mathsf{DEC}(\mathsf{ct}, \vec{id})$ where $\mathsf{ct} = (\mathsf{ct}', \pi, \vec{id}')$, the challenger calls $m \leftarrow \mathsf{Ext}((\vec{id}, \mathsf{ct}'), \pi, \mathsf{ek})$ and returns m.

By the true simulation f-extractability of Π_n, it follows that

$$|\Pr[S_1] - \Pr[S_2]| \leq \epsilon_f$$

Finally, note that **G2** can be perfectly simulated by an $\mathsf{LR\text{-}OW\text{-}CPA}$ adversary since in particular the decryption oracle can be simulated locally, and all other queries can be simulated using a combination of local computation and oracle queries. Thus we have

$$\Pr[S_2] = \epsilon_{cpa}$$

The result follows by collecting the probabilities. □

E.4 LR-KUOWR-secure kuKEM from LR-OW-CCA-secure HIBE

We provide in Fig. 15 a construction of kuKEM from HIBE provided by Balli, Rösler and Vaudenay [BRV20a] adapted to our notation. In particular, although [BRV20a] construct kuKEM from a HIB-KEM, or a hierarchical identity-based key encapsulation mechanism, the construction is essentially the same. Balli, Rösler and Vaudenay argue that their kuKEM is KUOWR-secure with a suitable $\mathsf{OW\text{-}CCA}$ security notion for HIB-KEM but do not provide a formal definition. We argue below security in our case given that both security notions consider leakage.

Theorem 5. *Let Π_h be a $(q, \ell_{\mathsf{mk}}, \ell_{\mathsf{sk}}, t, \epsilon_{cca})$-$\mathsf{LR\text{-}OW\text{-}CCA}$-secure HIBE. Then Π_k (Fig. 10) is $(q, t', q^2 \cdot \epsilon_{cca})$-$\mathsf{LR\text{-}KUOWR}$-secure for $t \approx t'$.*

Proof. We construct a number of $\mathsf{LR\text{-}OW\text{-}CCA}$ adversaries that simulates for $\mathsf{LR\text{-}KUOWR}$ adversary $\mathcal{A}$. Let $\mathcal{A}_{i,j} = \mathcal{A}'$ be an adversary that simulates for $\mathcal{A}$ given $\mathcal{A}$'s i-th query to SOLVE is winning, and it is with first argument tr that is the result of j queries to a combination of ENC and UPPK. Then, $\mathcal{A}'$ simulates $\mathsf{ENC}(r)$ and DEC by local simulation except for the ENC and DEC calls corresponding to tr. Leakage is simulated directly via the leakage oracle. Let tr' be the value of tr_{A} after $\mathcal{A}$ has made $j-1$-th query to $\mathsf{ENC}/\mathsf{UPPK}$ (note $j = 0$ is possible). At this point, $\mathcal{A}'$ calls $\mathsf{CHALL\text{-}OW}(\mathsf{tr}')$ and receives challenge ciphertext ct^* as output. Then, if $\mathcal{A}$'s j-th call is to UPPK, $\mathcal{A}'$ aborts, otherwise $\mathcal{A}'$ simulates $\mathsf{ENC}(r)$ via ct^* (aborting also if $r \neq \perp$. $\mathcal{A}'$ simulates $\mathsf{EXP}(\mathsf{tr})$ using $\mathsf{EXP\text{-}KEY}$. $\mathcal{A}'$ simulates the first $i-1$ calls to SOLVE by returning $\perp$. When $\mathcal{A}'$ makes its i-th query with input $(\mathsf{tr}', \mathsf{k})$, $\mathcal{A}$ returns k to its challenger. The simulation is perfect and the result follows by taking the union bound over all q^2 possible adversaries. □

```
Function Setup($1^\lambda$)
    $\mathsf{pp} \leftarrow 1^\lambda$
    return $\mathsf{pp}$

Function Gen(pp)
    $(\mathsf{pp}', \mathsf{sk}') \leftarrow \Pi_h.\mathsf{Setup}(\mathsf{pp})$
    $\vec{id} \leftarrow \epsilon$
    $\mathsf{pk} \leftarrow (\mathsf{pp}', \vec{id})$
    $\mathsf{sk} \leftarrow (\mathsf{pp}', \mathsf{sk}', \vec{id})$
    return $(\mathsf{pk}, \mathsf{sk})$

Function Encaps(pk)
    $(\mathsf{pp}', \vec{id}) \leftarrow \mathsf{pk}$
    $\mathsf{k} \leftarrow\!\$\ \mathcal{M}$
    $\mathsf{ct} \leftarrow \Pi_h.\mathsf{Enc}(\mathsf{pp}', k, id)$
    $id \leftarrow id \mid\mid \mathsf{ct}$
    $\mathsf{pk} \leftarrow (\mathsf{pp}', \vec{id})$
    return $(\mathsf{pk}, k, \mathsf{ct})$

Function Decaps(sk, ct)
    $(\mathsf{pp}', \mathsf{sk}', \vec{id}) \leftarrow \mathsf{sk}$
    $\mathsf{k} \leftarrow \Pi_h.\mathsf{Dec}(\mathsf{pp}', \mathsf{sk}', \mathsf{ct})$
    $\mathsf{sk}' \leftarrow \Pi_h.\mathsf{Del}(\vec{id}, \mathsf{sk}', \mathsf{ct})$
    $\vec{id} \leftarrow \vec{id} \mid\mid \mathsf{ct}$
    $\mathsf{sk} \leftarrow (\mathsf{pp}', \mathsf{sk}', \vec{id})$
    return $(\mathsf{sk}, \mathsf{k})$

Function UpPk(pk, ad)
    $(\mathsf{pp}', \vec{id}) \leftarrow \mathsf{pk}$
    $\vec{id} \leftarrow \vec{id} \mid\mid \mathsf{ad}$
    $\mathsf{pk} \leftarrow (\mathsf{pp}', \vec{id})$
    return $\mathsf{pk}$

Function UpSk(sk, ad)
    $(\mathsf{pp}', \mathsf{sk}', \vec{id}) \leftarrow \mathsf{sk}$
    $\mathsf{sk}' \leftarrow \Pi_h.\mathsf{Del}(\mathsf{pp}', \vec{id}, \mathsf{sk}', \mathsf{ct})$
    $\vec{id} \leftarrow \vec{id} \mid\mid \mathsf{ad}$
    $\mathsf{sk} \leftarrow (\mathsf{pp}', \mathsf{sk}', \vec{id})$
    return $\mathsf{sk}$
```

Fig. 15: Construction Π_k of LR-KUOWR-secure kuKEM from LR-OW-CCA-secure HIBE Π_h.

F Proof of Theorem 4

Our proof strategy is similar to that of Balli, Rösler and Vaudenay [BRV20a, Section 6], adapted to our notation, notions and construction.

Proof. We proceed by a hybrid argument and construct games **G0** to **G3**. Let $\Pr[S_i]$ be the probability that **Gi** outputs 1 for $i \in [0, 3]$.

G0: This is the LR-KUOWR game instantiated with Π_k. We have:

$$\Pr[S_0] = \Pr[\mathsf{LR\text{-}KUOWR}^{\ell}_{\mathcal{A}}(1^\lambda) \Rightarrow 1]$$

G1: This is the same as **G0**, except that for all outputs (k, k_m) and $(\tilde{k}, \tilde{k}_m$ of H, we have $k \neq \tilde{k}$. This follows from a standard birthday bound argument, and thus we have

$$|\Pr[S_0] - \Pr[S_1]| \leq \frac{q^2}{|\mathcal{K}|}$$

G2: This is the same as **G1**, except that:

– The adversary is given integer $g \in [0, q-1]$ as input such that $\mathsf{SOLVE}(\mathsf{tr}, \cdot)$ first outputs 1 (or $g = 0$ if no such query exists) where $\mathsf{tr} = \mathsf{tr}_\mathsf{A}$ after g queries to ENC.
– The experiment aborts if, *before* the g-th query to ENC is made as above, $\mathsf{DEC}(\mathsf{ct})$ was queried after $g-1$ in-sync queries to DEC were made with input $\mathsf{ct} = (\mathsf{ck}, \dots)$ and (the same) ck was sampled inside of Encaps in the g-th ENC call (made after).

Note first that are at most $q-1$ such ENC queries. Note also that the second event occurs with probability $1/|\mathcal{K}|$ (since ENC cannot be called with adversarial randomness). It follows from a standard argument that

$$\Pr[S_1] \leq q \cdot \left(\Pr[S_2] + \frac{1}{|\mathcal{K}|} \right)$$

<u>**G3**:</u> This is the same as **G2**, except that the game aborts if the adversary ever makes query $H(k')$, where k' is the output of the challenger's call to H inside of Encaps inside of the adversary's g-th call to ENC; let E be this event. One can construct an $\mathsf{LR\text{-}OW}$ adversary $\mathcal{A}'$ that simulates for **G2** adversary $\mathcal{A}$ wins given E. $\mathcal{A}'$ simulates:

– The challenger's input $(\mathsf{pp}, \mathsf{pk})$ to $\mathcal{A}$ using pp from its $\mathsf{LR\text{-}OW}$ challenger, calling $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \epsilon)$ and otherwise simulating locally. Note that this and subsequent $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \cdot)$ queries do not invalidate the simulation.
– $\mathsf{ENC}(\mathsf{ad}; \mathsf{r})$ via first a call to $\mathsf{SEND}((1, \mathsf{ck}), \mathsf{r})$ for simulated ck, followed by a call to $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \mathsf{tr})$ for the resulting tr, followed by $\mathsf{SEND}((2, \mathsf{ct}), 0)$ for the resulting ct.
– DEC via calls to $\mathsf{RECEIVE}$ (as well as via $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \cdot)$ and then local simulation for the Send query if not already done above).
– $\mathsf{UPPK}(\mathsf{ad})$ via $\mathsf{SEND}(\mathsf{A}, (0, \mathsf{ad}), 0)$ and then $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \cdot)$.
– UPSK via local simulation (first $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \cdot)$ if not previously called) and $\mathsf{RECEIVE}((0, \mathsf{ad}), \mathsf{ct})$.
– $\mathsf{EXP}(\mathsf{tr})$ via $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \mathsf{tr}')$ (if necessary) and $\mathsf{EXP\text{-}STATE}(\mathsf{B}, \mathsf{tr}')$ for the corresponding tr'.
– $\mathsf{SOLVE}(\mathsf{tr}, k)$ by outputting $\perp$ if tr does not correspond to the g-th ENC call, and otherwise by checking if there exists k' such that $H(k') = (k, k_m)$, and calling $\mathsf{CHALL\text{-}OW}(\mathsf{A}, \mathsf{tr}, k')$ for such a k' (which is unique if it exists by definition of **G1**).
– $\mathsf{LEAKSK}(f, \mathcal{P}, \mathsf{tr})$ as follows. Recall $f = f(\mathsf{st}_\mathsf{A}, \mathsf{st}_\mathsf{B})$ is a function of both st_A and st_B. $\mathcal{A}'$ therefore first calls $\mathsf{EXP\text{-}STATE}(\mathsf{A}, \mathsf{tr}')$ for the corresponding tr' (if not yet done already) to obtain st_A. $\mathcal{A}'$ then calls $\mathsf{LEAK\text{-}STATE}(f_{\mathsf{st}_\mathsf{A}}, \mathsf{B}, \mathsf{tr}')$ where $f_{\mathsf{st}_\mathsf{A}}$ is f evaluated on input st_A. Note that this exposure of A's state is allowed since the g-th ciphertext (if it exists) is such that ENC must have been called with honest randomness, and full exposure of all other keys is permitted since they are not challenges.

Recall that only keys derived in ENC queries can be guessed by the adversary, unlike in URKE where keys output by an out-of-sync receiver can be challenged.

– $H(k')$ by lazily sampling. In addition, if $\mathcal{A}$ has not yet made their g-th query to ENC, $\mathcal{A}$ stores k'. Once soon as $\mathcal{A}$ has made their g-th query to ENC, letting tr^* be the resulting value of tr_A, let tr_k be the corresponding kuKEM trace after the first Send query in the corresponding Encaps call. Then for all stored k', $\mathcal{A}'$ calls $\mathsf{SOLVE}(\mathsf{tr}_k, k')$, and likewise for each subsequent $H(k')$ query. If $H(k')$ ever outputs 1, $\mathcal{A}'$ stops simulating and finishes executing.

The simulation of **G2** is perfect given $\neg E$, and given E, some query to SOLVE that $\mathcal{A}'$ makes will result in output 1. Note $\mathcal{A}'$ makes at most 3 queries to its oracles for every query $\mathcal{A}$ makes. By standard bad event analysis it then follows that

$$|\Pr[S_2] - \Pr[S_3]| \leq \epsilon_u$$

Now by definition of **G3**, $(k, k_m) \leftarrow H(k')$ is not queried by the adversary where $H(k')$ is called by the challenger in the g-th ENC call. Therefore modulo leakage k_m is uniform to $\mathcal{A}$. **G3** can then be perfectly simulated by an $\mathsf{LR\text{-}OT\text{-}SUF}$ adversary, the argument following that of [BRV20a] except that **G3** leakage is additionally simulated via $\mathsf{LR\text{-}OT\text{-}SUF}$ oracle $\mathsf{LEAK\text{-}KEY}$ (where in the argument, the $\mathsf{LR\text{-}OT\text{-}SUF}$ MAC key is embedded in the simulation of the g-th ENC call), except for $g = 0$ the simulator can simply abort. We thus arrive at

$$\Pr[S_3] = \epsilon_m$$

The result follows by collecting the probabilities. □