An extended abstract [DHRR22a] of this article appears in the proceedings of ASIACRYPT 2022. The full version [DHRR22b] of this article is available in the IACR eprint archive.

Strongly Anonymous Ratcheted Key Exchange

Benjamin Dowling[1], Eduard Hauck[2], Doreen Riepel[2], and Paul Rösler[3]

[1] University of Sheffield b.dowling@sheffield.ac.uk
[2] Ruhr-Universität Bochum {eduard.hauck,doreen.riepel}@rub.de
[3] New York University paul.roesler@cs.nyu.edu

Abstract. Anonymity is an (abstract) security goal that is especially important to threatened user groups. Therefore, widely deployed communication protocols implement various measures to hide different types of information (i.e., metadata) about their users. Before actually defining anonymity, we consider an attack vector about which targeted user groups can feel concerned: continuous, temporary exposure of their secrets. Examples for this attack vector include intentionally planted viruses on victims' devices, as well as physical access when their users are detained.

Inspired by *Signal's Double-Ratchet Algorithm*, *Ratcheted* (or *Continuous*) *Key Exchange* (RKE) is a novel class of protocols that increase *confidentiality* and *authenticity* guarantees against temporary exposure of user secrets. For this, an RKE regularly renews user secrets such that the damage due to past and future exposures is minimized; this is called *Post-Compromise Security* and *Forward-Secrecy*, respectively.

With this work, we are the first to leverage the strength of RKE for achieving strong *anonymity* guarantees under temporary exposure of user secrets. We extend existing definitions for RKE to capture attacks that interrelate ciphertexts, seen on the network, with secrets, exposed from users' devices. Although, at first glance, strong authenticity (and confidentiality) conflicts with strong anonymity, our anonymity definition is as strong as possible without diminishing other goals.

We build strongly anonymity-, authenticity-, and confidentiality-preserving RKE and, along the way, develop new tools with applicability beyond our specific use-case: *Updatable and Randomizable Signatures* as well as *Updatable and Randomizable Public Key Encryption*. For both new primitives, we build efficient constructions.

Keywords: RKE, CKE, Ratcheted Key Exchange, Continuous Key Exchange, Anonymity, Secure Messaging, State Exposure, Post-Compromise Security

1 Introduction

Anonymity. Traditionally, anonymity means that participants of a session cannot be *identified*. As we will argue below, this notion of anonymity is very narrow. Furthermore, in the context of this work, it is not immediately clear what the identity of a session participant actually is. The reason for this is that we consider a modular protocol stack that consists of a *Session Initialization Protocol* (SIP; e.g., an authenticated key exchange) and an independent, subsequent *Session Protocol* (SP; e.g., a symmetric channel or a ratcheted key exchange). According to this modular composition paradigm, only the SIP actually deals with users and their identities, and groups them into session participants who execute the subsequent SP. While the SP may assign different roles to its session participants, the SP is (usually) agnostic about their identities. Thus, it cannot reveal identities by definition. Nevertheless, the context of an SP session and the role of its participant therein may suffice to identify the underlying identity.

Session Protocols. In this work, we focus on anonymity for SPs. Roughly, we call an SP *anonymity-preserving* if its execution reveals nothing about its context, including the session participants, the protocol session itself, the status of a session, etc. We note that real-world deployment of an anonymity-preserving SP requires more than that—e.g., an anonymous SIP, a delivery protocol that transmits anonymous traffic across the Internet, or a mechanism that ensures a large enough set of potential protocol users. While these external components are outside the scope of our work, we mind the broader execution environment of SPs to direct our definitions.

Exposure of Secrets. Intuitively, anonymity complements standard security goals, such as confidentiality and authenticity, by requiring that *publicly observable* context data (or *metadata*) remains hidden.

More specifically, anonymity means that ciphertexts on the network cannot be interrelated. In this work, we augment this perspective by considering adversaries against anonymity who can expose information that is *secretly stored* by the targeted users. Consequently, our notion of anonymity requires that it is hard to interrelate these exposed user secrets with publicly visible data.

Temporary exposure of user secrets is a realistic threat, especially against cryptographic protocols with long-lasting sessions. The most prominent example for this type of long-term protocols is secure messaging where sessions almost never terminate and, hence, can last for several years. Therefore, anticipating the exposure of participants' locally stored secrets during the lifetime of a session is advisable.

Ratcheted Key Exchange. Inspired by Signal's Double-Ratchet Algorithm [PM16], *Ratcheted Key Exchange* (RKE) is an SP primitive that provides security in the presence of adversaries who can expose session participants' local secrets. The core idea of RKE is that the participants continuously establish new symmetric session keys. Following the modular composition paradigm, these keys can be used by another subsequent SP, for instance, to encrypt payload data symmetrically. While establishing session keys, the participants update and renew all their local secrets to recover from potential past exposures (Post-Compromise Security; PCS), and delete old secrets before a potential future exposure occurs (Forward-Secrecy; FS). So far, RKE was only used for preserving *secrecy* and *authenticity* of session keys under the exposure of secrets. In order to also achieve strong *anonymity* under exposure of secrets, we are the first to take advantage of RKE.

Examining RKE constructions, one may doubt that this secrecy- and authenticity-preserving primitive can be extended to also realize strong anonymity: On the one hand, authenticity and anonymity generally tend to be incompatible security goals. On the other hand, for continuously performing updates, participants locally store structured information that is often encoded in sent and received ciphertexts, or has traceable relations to the secrets stored by other session participants. Avoiding this structure (and hiding all relations between sender secrets, ciphertexts, and receiver secrets) is highly non-trivial.

We start with extending RKE syntactically to account for an environment in which preserving anonymity is crucial. Then, we specify a security definition that captures strong anonymity under exposure of secrets. This new definition is compatible with strong secrecy and authenticity notions of RKE.

Flavors of RKE. To reduce complexity and maintain clarity, we consider *unidirectional* RKE [BSJ+17, PR18b, BRV20], which is a simple, natural notion of RKE that restricts communication between two session participants, Alice and Bob, to flow only from the former to the latter. We leave it an open, highly non-trivial[4] problem for future work to extend our results to more complex *bidirectional* RKE (e.g., [PR18b, JS18, PR18a]), RKE with *immediate decryption* (e.g., [ACD19]), RKE in *static* groups (e.g., [CCG+18]) and *dynamic* groups (e.g., [RMS18, ACDT20, BDG+22]), resilient to *concurrent* operations (e.g., [BDR20, AAN+22]), etc. In Appendix G, we take a look at the "unidirectional core" of each *two-party* RKE construction from the literature and present successful attacks against anonymity for all of them. We refrain from also presenting (non-trivial) attacks against constructions from the *group* setting without having a suitable anonymity definition that formally separates *trivial* attacks from *non-trivial* ones.[5]

Further Related Work. The literature of anonymity-preserving cryptography ranges from key-private public key encryption (e.g., [BBDP01, KMO+13, GMP22]) to anonymous signatures (e.g., [YWDW06, Fis07]) to privacy-preserving key exchange (e.g., [Zha16, SSL20, IY22]) to anonymous onion encryption (e.g., [DS18, RZ18a]) and many other primitives. In principle, our definitions are in line with these notions insofar that we require indistinguishability of "everything that the adversary sees" for a real RKE execution (i.e., ciphertexts and exposed user secrets) from independently sampled equivalents. While some previous works furthermore cover non-cryptographic properties such as anonymous delivery mechanisms (see, e.g., [DS18]), our work abstracts these external components. To the best of our knowledge, anonymity under (temporary and continuous) exposure of user secrets has not been formally studied before.

[4] Immediate extension and generalization of our results seems unlikely, given the remarkable gap of complexity between non-anonymous unidirectional RKE and more advanced non-anonymous types of RKE.

[5] Note that all CGKA (or "group RKE") constructions reveal structural information like the group size via (publicly) sent ciphertexts. (Moreover, these constructions let users store information about other members in the local user states, and most constructions rely on an active server that participates in the protocol execution.) However, without a formal, satisfiable anonymity definition, it is unclear which information can theoretically be hidden, even by an ideal CGKA construction.

Nevertheless, anonymity, privacy, and deniability is generally considered relevant in the domain of secure messaging. For example, the Signal messenger implements the Sealed Sender mechanism [Sig18] to hide the identities of senders. Yet, this mechanism is stateless and uses static long-term secrets, which means that it is insecure under the exposure of receiver secrets. Besides this, several attacks against the deployment of Sealed Sender [MKA+21, TLMR22] undermine its anonymity guarantees. The Sealed Sender mechanism is related to instances of the Noise protocol framework [Per18, DRS20] that also claims to reach various notions of anonymity. Yet, the established symmetric session key in a Noise protocol session is static, which means that its exposure breaks anonymity, too. Finally, there is an ongoing discussion about privacy and deniability in the MLS standardization initiative [BBR+22] that is yet to be concluded.[6] Related to this, Emura et al. [EKN+22] informally propose changes to an early version of MLS by Cohn-Gordon et al. [CCG+18] in order to hide the identities of group members. As mentioned above, this is a rather weak form of anonymity. Finally, we note that none of our definitions requires deniability and none of our constructions reaches deniability.

Contributions. Our main contributions are defining *anonymity* for *Ratcheted Key Exchange* (RKE) and designing a construction that provably satisfies this definition. However, we do not naïvely adopt and extend prior notions of RKE, but we take a fresh look at this primitive, keeping in mind the overall execution environment in which anonymity is important.

Along the way, we develop two new tools that we use to build our final RKE construction. The first tool, *Updatable and Randomizable Public Key Encryption* (urPKE), realizes anonymous PKE with randomizable encryption keys and updatable key pairs. We believe this has applications beyond our work, for example, to Updatable PKE [JMM19a, ACDT20, DKW21]. The second tool, *Updatable and Randomizable Signatures* (urSIG), simultaneously provides strong anonymity and authenticity guarantees. Roughly, it achieves strong unforgeability of signatures if the signing key is uncorrupted. Furthermore, the signer can derive multiple signing keys that work for the same verification key. However, it should be hard to derive the verification key from a signing key and, beyond that, hard to distinguish whether two signing keys correspond to the same verification key. Surprisingly, both urPKE and urSIG can be built efficiently from cryptographic standard components.

We focus on *anonymity* of RKE and its building blocks in the main body of this paper. All novel definitions, constructions, and proofs regarding other security goals such as authenticity and secrecy (which are valuable contributions), are summarized in the subsequent technical overview (Section 1.1). The full details of these summarized results can be found in the appendix.

1.1 Technical Overview

Unidirectional Ratcheted Key Exchange. Definitions and constructions of *Ratcheted Key Exchange* (RKE) in the literature are highly complex. Since we are the first to consider *anonymity* for this primitive, we want to focus on the core challenges that arise due to the interplay of strong anonymity, confidentiality, and authenticity. Furthermore, we present novel, insightful solutions for these challenges. Thus, for didactic reasons, we condense the question of how to define and construct anonymous RKE by considering the simplest variant of this primitive—so called *Unidirectional* RKE (URKE) [BSJ+17, PR18b, BRV20]. As we will see, definitions and constructions of anonymous RKE become complex even for this simple unidirectional variant.

An RKE session between two users begins with the initialization that produces a secret state for each user $\mathsf{RKE.init} \to_{\$} (\mathrm{stS}, \mathrm{stR})$. (In practice, this abstract initialization can be instantiated by using an authenticated key exchange protocol.) The users then continuously use their secret states to asynchronously send ciphertexts to their partners. These ciphertexts establish fresh symmetric keys (for the use in subsequent, higher layer SPs) and refresh the secrets in both users' states. While a fully *bidirectional* RKE scheme allows both users to establish new symmetric keys, a *unidirectional* RKE scheme assigns different roles to the two users: only one user (Alice) sends ciphertexts to establish new keys $\mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad}) \to_{\$} (\mathrm{stS}, c, k)$ and the other user (Bob) receives these ciphertexts to compute these (same) established keys $\mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad}) \to_{\$} (\mathrm{stR}, k)$. Either way, secrets in both users' states are continuously renewed by these operations.

[6] See the discussion thread initiated here: https://mailarchive.ietf.org/arch/msg/mls/-1VF95d8od0lF_AFj2WMvk5SQXE/.

Standard Security Goals. Secrecy and authenticity of established symmetric keys for URKE have been studied in prior work [BSJ+17,PR18b,BRV20]. These works extend standard secrecy and authenticity notions by allowing the adversary to expose the secret states of Alice and Bob before *and* after each of their send and receive operations, respectively.

Key Secrecy. For *secrecy* of URKE [PR18b], we require that all symmetric keys established by Alice are indistinguishable from random keys unless Bob's corresponding secret state was exposed earlier. More precisely, the symmetric key established by Alice's i_k-th ciphertext must be secure, unless Bob's secret state was exposed already after successfully processing the first i_x ciphertexts from Alice, where $i_\mathrm{x} < i_\mathrm{k}$. By correctness, Bob's (exposed) state after processing Alice's first i_x ciphertexts can always be used to successfully process the subsequent $i_\mathrm{k} - i_\mathrm{x}$ ciphertexts from Alice and then compute the i_k-th symmetric key. This notion captures *post-compromise security* (PCS) and *forward-secrecy* (FS) on Alice's side, since all her established symmetric keys must remain secure independent of whether her secret state is ever exposed. It also captures a strong notion of FS on Bob's side, since exposures of his state must not impact the secrecy of a key established with ciphertext i_k under two conditions: (1) the exposures occurred after Bob received ciphertext i'_x, and $i_\mathrm{k} \leq i'_\mathrm{x}$, or (2) Bob falsely accepted an earlier ciphertext $i_\mathrm{f}, i_\mathrm{f} < i_\mathrm{k}$ that was not sent by Alice and Bob was exposed subsequently at point i'_x, and $i_\mathrm{f} \leq i'_\mathrm{x}$. This requires that Bob's state becomes incompatible with Alice's state immediately after accepting a forged ciphertext.

Authenticity. Authenticity for URKE [DV19] a ciphertext i_f, unless Alice's matching secret state was exposed. More precisely, after successfully accepting $i_\mathrm{f} - 1$ ciphertexts from Alice, Bob must reject the i_f-th ciphertext if it was not sent by Alice, unless Alice's secret state was exposed after sending the i_x-th ciphertext, where $i_\mathrm{x} = i_\mathrm{f} - 1$. We call such a successful trivial ciphertext forgery a *trivial impersonation*.

Robustness and Recover Security. We consider two additional properties for URKE: *robustness* and *recover security*. The former requires that Bob will not change his state when rejecting a ciphertext. Thus, Bob can uphold his communication with Alice even if he sometimes receives (and rejects) false ciphertexts that did not result in a trivial impersonation. When considering (receiver) anonymity, robustness is a valuable feature as it allows Bob to perform "trial decryptions" to check if a ciphertext was meant for him or not. Furthermore, consider a setting in which Bob is the receiver of many independent URKE sessions. Due to (sender) anonymity, he may not know the sender of a ciphertext, so he can "trial decrypt" the ciphertext with all of his receiver states until one of them accepts. We conclude that robustness is a crucial property for anonymous RKE. Recover security [DV19] requires that, whenever Bob falsely accepts a trivial impersonation ciphertext, he will never again accept a ciphertext sent by Alice. This ensures that an adversary who conducted a successful trivial impersonation cannot hide this attack by letting Alice and Bob resume their communication.

For comprehensibility, we make the simplifying assumption that Alice always samples "good" randomness for her send operations. While "bad" randomness can be a realistic threat in some scenarios, we note that URKE under bad randomness—beyond causing more complex definitions and constructions—must rely on strong and inefficient HIBE-like building blocks as Balli et al. [BRV20] prove. We leave it an open problem to extend our results to stronger threat models.

Known Constructions. RKE constructions only achieving the above properties can be built from standard public key encryption (PKE) and one-time signatures (OTS) [PR18b,JS18,DV19]. The idea is that Alice (1) generates fresh PKE key pair $(\mathrm{ek}_i, \mathrm{dk}_i)$ and OTS key pair $(\mathrm{vk}_i, \mathrm{sk}_i)$ with every send operation i. She then (2) encrypts the new decryption key dk_i with the prior encryption key ek_{i-1}, and she (3) signs the resulting PKE ciphertext as well as the new verification key vk_i with the prior signing key sk_{i-1}. The composed URKE ciphertext consists of PKE ciphertext, new verification key, and signature. Alice deletes all prior values as well as the new decryption key dk_i and sends the composed URKE ciphertext to Bob, who verifies the signature, decrypts the PKE ciphertext, and stores $(\mathrm{dk}_i, \mathrm{vk}_i)$. An additional hash-chain over the entire sent (resp. received) transcript maintains consistency between Alice and Bob, and additional encrypted key material sent from Alice to Bob establishes the symmetric session keys.

Shortcomings. To understand why the above construction does not provide anonymity, note that standard (one-time) signatures can reveal the corresponding verification key. Thus, it can be easy to link two subsequent URKE ciphertexts by testing whether the signature contained in one ciphertext verifies under the verification key contained in the other. (More detailed attacks against anonymity of existing two-party RKE constructions are in Appendix G.) To overcome this limitation, one could simply encrypt the verification key along with the transmitted decryption key. This prevents adversaries who only see ciphertexts transmitted on the network from linking these ciphertexts and, thereby, attributing them to the same URKE session. As we will argue next, this weak level of anonymity is inadequate for settings in which ratcheted key exchange is deployed.

Defining (Strong) Anonymity. The main goal of ratcheted key exchange is to continuously establish symmetric keys that remain secure even if the involved users' secret states are temporarily exposed earlier (PCS) and/or later (FS). Hence, if temporary state exposure is considered a realistic threat against secrecy of keys, it is also a realistic threat against anonymity. Consequently, we allow an adversary against anonymity to expose both Alice's and Bob's states.

Ciphertext Anonymity. In a first attempt to define anonymity, we follow the standard concept from the literature: We require that ciphertexts sent from Alice to Bob cannot be distinguished from ciphertexts sent in an independent URKE session from Clara to David, even if the adversary can expose Alice's and Bob's secret states. In this preliminary notion that we call *ciphertext anonymity*, adversaries can perform a trivial exposure that we have to forbid in order to obtain a sound definition. Forbidding this attack, ciphertext anonymity requires that Alice's i_c-th ciphertext must be indistinguishable from a ciphertext sent in an independent URKE session, unless Bob's secret state was exposed already after successfully processing the first i_x ciphertexts from Alice, where $i_\mathrm{x} < i_\mathrm{c}$. Note that by authenticity, Bob's (exposed) state after processing Alice's first i_x ciphertexts can always be used to verify whether the subsequent $i_\mathrm{c} - i_\mathrm{x}$ ciphertexts were sent by Alice or by an independent user. This notion captures *post-compromise anonymity* (PCA) and *forward-anonymity* (FA) on Alice's side, since all her ciphertexts must remain anonymous independent of whether her secret state is ever exposed. It also captures a strong notion of FA on Bob's side, since exposures of his state must remain harmless for the anonymity of a ciphertext i_c under two conditions: (1) the exposures were conducted after Bob received ciphertext i'_x, and $i_\mathrm{c} \leq i'_\mathrm{x}$, or (2) Alice was trivially impersonated towards Bob with an earlier ciphertext i_f, and $i_\mathrm{f} < i_\mathrm{c}$ and Bob was exposed after ciphertext i'_x, and $i_\mathrm{f} \leq i'_\mathrm{x}$.

Full Anonymity. Our above description of ciphertext anonymity is not fully formal and the attentive reader may have identified a gap. Consider an adversary who exposes Alice's state twice, once before seeing a ciphertext on the network and once afterwards. By only checking if Alice's state changed between these exposures, the adversary can determine if the ciphertext was sent by Alice. (Note that by authenticity, Alice's state must change with every send operation whereas the state does not change as long as Alice remains inactive.)

To mitigate the threat that Alice's exposed URKE states reveal whether she sent something, we extend the syntax of URKE by adding algorithm $\mathsf{RKE.rr}(\mathrm{stS}) \rightarrow_\$ \mathrm{stS}$ that (re-)randomizes her state on demand. Executing this algorithm between two exposures, Alice's state can be changed independent of whether she sent a ciphertext. Thus, she can hide if she was the sender of a ciphertext that the adversary observed.

Before specifying a corresponding (stronger) notion of anonymity, we present another threat against anonymity. Consider an adversary who can observe all URKE ciphertexts sent from Alice's device. At some point, this adversary exposes all secrets Alice stores on her device. If Alice has only one stored URKE state, the adversary knows that all observed URKE ciphertexts were sent with this state in the same single session. Since Alice may want to hide how many URKE sessions are running on her device, and how many URKE ciphertexts are sent in each of these sessions, she may want to set up "dummy" URKE states. This scenario motivates that we require for anonymity that Alice's and Bob's secret states must be indistinguishable from independent secret sender and receiver states, respectively—beyond requiring that ciphertexts between Alice and Bob must be indistinguishable from ciphertexts sent in an independent session.

In summary, we require that all secret states that an adversary exposes and all ciphertexts that an adversary observes on a network must be indistinguishable from independent secret states and ciphertexts, respectively, unless correctness, secrecy, and authenticity impose conditions that inevitably allow for distinguishing them. This notion of anonymity is extremely strong and its precise pseudo-code definition is rather complex. However, the basic concept is relatively simple.

Security Experiment. An adversary $\mathcal{A}$ against anonymity plays a game in which it has adaptive access to the following oracles: $\texttt{Snd}$, $\texttt{RR}$, $\texttt{Rcv}$, $\texttt{Expose}_S$, $\texttt{Expose}_R$. Internally, these oracles execute Alice's $\mathsf{RKE.snd}$ algorithm, outputting the resulting ciphertext, Alice's $\mathsf{RKE.rr}$ algorithm, Bob's $\mathsf{RKE.rcv}$ algorithm, and expose Alice's and Bob's current secret states stS and stR, respectively. Access to these oracles is standard in the literature on RKE (except for oracle $\texttt{RR}$ for the additional $\mathsf{RKE.rr}$ algorithm). In addition, the adversary can adaptively query oracles that depend on a challenge bit b that is randomly sampled at the beginning of the game:

- $\texttt{ChallSnd}$ equals oracle $\texttt{Snd}$ iff $b = 0$; otherwise, it temporarily initializes a new, independent URKE session with algorithm $\mathsf{RKE.init}$, uses the temporary sender to send a ciphertext with algorithm $\mathsf{RKE.snd}$, and outputs this ciphertext (the temporary URKE session is discarded immediately afterwards); oracle $\texttt{Rcv}$ silently ignores ciphertexts created by $\texttt{ChallSnd}$ under $b = 1$
- $\texttt{ChallExpose}_S$ equals oracle $\texttt{Expose}_S$ iff $b = 0$; otherwise, it initializes a new, independent session with algorithm $\mathsf{RKE.init}$ (as above) and outputs the resulting secret sender state
- $\texttt{ChallExpose}_R$ equals oracle $\texttt{Expose}_R$ iff $b = 0$; otherwise, it behaves as oracle $\texttt{ChallExpose}_S$ under $b = 1$, except that it outputs the resulting temporary secret receiver state

The adversary wins the game if it determines challenge bit b without performing a trivial attack that inevitably reveals this challenge bit.

Identifying Trivial Attacks. To complete the above anonymity definition, all attacks that trivially reveal the challenge bit have to be identified, detected, and forbidden. Our aim is to detect these attacks as precisely as possible such that the restrictions limit the adversary as little as possible (leading to a strong definition of anonymity). Interestingly, one class of trivial attacks is particularly hard to detect in a precise way for the anonymity game: trivial impersonations. To give a simple, clarifying example for this, we consider the following adversarial schedule of oracle queries: (1) $\texttt{ChallExpose}_S \to \mathrm{stS}_b$, (2) $\texttt{Rcv}(c')$, where c' is crafted by the adversary[7], (3) $\texttt{Expose}_R \to \mathrm{stR}$.

We begin with the case $b = 1$, which means that the adversary plays in the random world. In this world, exposed state $\mathrm{stS}_b = \mathrm{stS}_1$ is a random sender state that corresponds to a hidden temporary receiver state independent of Bob's actual receiver state stR at step (1). Thus, by authenticity, Bob should not accept any adversarially crafted ciphertext c' in this case. Put differently, impersonating Alice towards Bob is non-trivial for this adversarial behavior in the random world. This means that Bob will reject c' with high probability and the exposed receiver state of Bob in step (3) remains stR, which is independent of the sender state stS_1 exposed in step (1).

In contrast, if $b = 0$, which means that the adversary plays in the real world, exposed sender state $\mathrm{stS}_b = \mathrm{stS}_0$ corresponds to the real receiver state of Bob stR at step (1). Hence, stS_0 can be used to craft a valid ciphertext forgery c' that trivially impersonates Alice towards Bob. If the adversary, indeed, performs such a trivial impersonation by executing $\mathsf{RKE.snd}(\mathrm{stS}_0) \to_\$ (\mathrm{stS}', c', k')$ and querying $\texttt{Rcv}(c')$, Bob will compute $\mathsf{RKE.rcv}(\mathrm{stR}, c') \to (\mathrm{stR}', k')$.[7] The state of Bob stR' that is exposed in final step (3) corresponds to the state stS' that the adversary computed (in their head) during the impersonation. By authenticity, a pair of corresponding states $(\mathrm{stS}', \mathrm{stR}')$ can always be identified as such by sending with the sender state and receiving the result with the receiver state.

Our full anonymity game must, consequently, forbid the final exposure in step (3) because otherwise the adversary can determine the challenge bit from the exposed state.

The presented trivial attack serves as the simplest example for multiple, more complicated trivial impersonations that our game must detect, which we describe in Section 4.2.

Main Components of Construction. At a first glance, our new URKE construction that fulfills the above anonymity notion follows the design principle of prior non-anonymous URKE constructions described earlier. That means intuitively, in every send operation, Alice (1) generates new PKE and OTS key pairs, (2) encrypts fresh secrets to Bob with which he can compute his matching new PKE decryption key (and the symmetric session key), and she (3) signs the resulting PKE ciphertext. Yet, the exact details of our construction are far more sophisticated. We proceed with presenting the most important anonymity requirements and the corresponding solutions implemented in our construction. A conceptual visualization of our construction is given in Figure 1.

[7] For simplicity, we ignore the associated data input ad here.

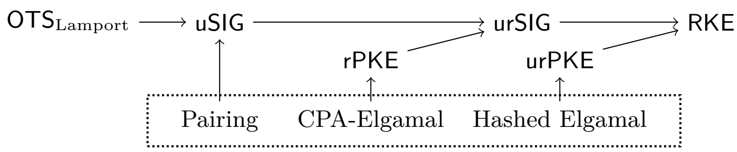

Fig. 1. Overview of URKE construction RKE from Updatable and Randomizable Signature urSIG as well as Updatable and Randomizable PKE urPKE with corresponding instantiations.

Hiding the Signature. Without presenting the full details of our anonymity definition yet, we note that it imposes the following intuitive requirements: (1) adversaries are allowed to see all (challenge) ciphertexts between sender and receiver; (2) seen (challenge) ciphertexts must remain anonymous even if Alice's state was ever exposed by the adversary before; (3) the authenticity notion presented above imposes the use of asymmetric authentication methods (i.e., signatures) from Alice to Bob. Thus, Alice must have a signing key stored in her state (due to (3)) that is potentially known by the adversary (due to (2)) and, simultaneously, her ciphertexts must be authenticated by corresponding signatures in an anonymous way (due to (1)+(2)+(3)). To ensure that the adversary cannot link matching signing keys (from Alice's exposed states) and signatures (in the sent ciphertexts), our construction encrypts signatures. This encryption of signatures is implemented deterministically with a symmetric key that is encrypted in the PKE ciphertext. Thus, the signature remains confidential while the signed ciphertext is determined before the signature is created, which maintains authenticity and anonymity.

Randomizing Signing Keys Anonymously. The second property required by our anonymity notion focuses on Alice's sender states before and after executing the RKE.rr algorithm. The two sender states of Alice, exposed before and after executing the RKE.rr algorithm, respectively, must be indistinguishable from two freshly generated, independent sender states. That means, an adversary must not learn whether the signing keys, stored in both states of Alice, produce signatures that are valid under the same verification key.[8] For this, we introduce the new notion of *Updatable and Randomizable Signatures* (urSIG) below.

Randomizing Encryption Keys Anonymously. Much like the relationship between two signing keys must be hidden by state randomizations, two PKE encryption keys, stored in Alice's exposed states, should not be easily linked. Namely, (a) encryption keys must look random, (b) there must be an routine that re-randomizes them, and (c) it cannot be determined which ciphertexts were created by them. For this, we introduce the new notion of *Updatable and Randomizable Public Key Encryption* (urPKE) below.

Updatable and Randomizable Public Key Encryption. We start with a high level overview of urPKE. As mentioned above, urPKE encryption keys must look random, be re-randomizable, and look independent of the ciphertexts that they produce. Our construction is based on ElGamal encryption. The encryption key consists of $\mathrm{ek} \leftarrow (g^r, g^{xr})$, where r and x are random exponents and $x = \mathrm{dk}$ is the decryption key. For re-randomizing the encryption key, we apply the same random exponent r' to both of its components $(\mathrm{ek}_0^{r'}, \mathrm{ek}_1^{r'})$. Encryption of message m takes a random exponent s to create ciphertext $c \leftarrow (\mathrm{ek}_0^s, \mathsf{H}(\mathrm{ek}_0^s, \mathrm{ek}_1^s) \oplus m)$. Decryption follows immediately via $m \leftarrow \mathsf{H}(c_0, c_0^{\mathrm{dk}}) \oplus c_1$.

This idea has applications beyond our specific use-case. For example, we point out how our construction can be extended to realize anonymous Updatable PKE [JMM19a, ACDT20, DKW21] that is broadly used in the literature of RKE and secure messaging.

Updatable and Randomizable Signatures. The security requirements for our new signature primitive urSIG are more challenging. Concretely, an urSIG scheme must provide the following properties: (a) verification keys must look random, (b) deriving the matching verification key from a signing key must be hard, and, beyond this, (c) determining whether two signing keys can produce signatures valid under the same (unknown) verification key must be hard. While ostensibly related to *Designated Verifier Signatures*, urSIG is a novel, incomparable primitive.

Construction Idea. Although the above requirements appear contradictory, we provide a simple construction. The idea is based on Lamport signatures [Lam79]. Intuitively, we start generating the signing key by sampling $2 \cdot \ell$ pre-images $\mathrm{sk}'_{i,b}, (i,b) \in [\ell] \times \{0,1\}$. To derive the matching verification key, we apply a one-way function on each pre-image $\mathrm{vk}'_{i,b} \leftarrow \mathrm{f}(\mathrm{sk}'_{i,b})$. Finally, we generate a PKE key pair $(\mathrm{ek}, \mathrm{dk})$ that

[8] Note that RKE.rr only randomizes Alice's state without any interaction with Bob.

allows ciphertext re-randomization. The final verification key consists of the decryption key dk and all images $\mathrm{vk}'_{i,b}$. The final signing key consists of the encrypted pre-images $\mathrm{sk}_{i,b} \leftarrow \mathsf{rPKE.enc}(\mathrm{ek}, \mathrm{sk}'_{i,b})$. To re-randomize Alice's verification key, she re-randomizes each component ciphertext $\mathrm{sk}_{i,b}$. The signature of message $m = (m_1, \ldots, m_\ell)$ consists of the respective signing key components $\sigma \leftarrow (\mathrm{sk}_{1,m_1}, \ldots, \mathrm{sk}_{\ell,m_\ell})$. To verify the signature, Bob decrypts each component and applies the one-way function for comparison with his verification-key component.

For strong unforgeability, we use a technique similar to the CHK transform [MRY04, CHK04] by employing a strongly unforgeable OTS that signs the actual message. The scheme above then signs the verification key of the strongly unforgeable OTS.

Shrinking Signatures. A drawback of this basic urSIG scheme is that it has large verification keys and large signatures. To mitigate the latter, we instantiate the above construction with a bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$ is the ciphertext space of the PKE scheme and $\mathbb{G}_2$ and $\mathbb{G}_T$ are chosen such that they are of sufficient size. This allows for aggregation of signing key components $(\mathrm{sk}_{1,m_1}, \ldots, \mathrm{sk}_{\ell,m_\ell})$ to obtain a compact signature σ; this aggregation is inspired by BLS signatures [BLS01, BGLS03]. The full details of this construction are in Section 6.

Performance. The computational and communication complexities of our overall RKE construction are dominated by the performance of the underlying urSIG construction. Generating a urSIG key is dominated by computing 2ℓ pairings, where ℓ can be considered the 'security parameter'. Signing keys consist of $4\ell + 1$ group elements in $\mathbb{G}_1$ and verification keys consist of 2ℓ group elements in $\mathbb{G}_T$ and a scalar in $\mathbb{Z}_p$; urSIG signing needs 4ℓ group operations in $\mathbb{G}_1$ to produce signatures of size 2 group elements in $\mathbb{G}_1$; urSIG verification is dominated by 2ℓ group operations in $\mathbb{G}_2$. This affects the RKE construction as follows: The computational complexity of $\mathsf{RKE.init}$ is dominated by sampling a urSIG key pair. The communication and computational complexities of $\mathsf{RKE.snd}$ are dominated by computing a urSIG signature and sending a urSIG verification key. The computational complexity of $\mathsf{RKE.rcv}$ is dominated by verifying an urSIG signature.

2 Preliminaries

We write $h \xleftarrow{\$} \mathcal{S}$ to denote that the variable h is uniformly sampled from finite set $\mathcal{S}$. For integers $N, M \in \mathbb{N}$, we define $[N, M] := \{N, N+1, \ldots, M\}$ (which is the empty set for $M < N$) and $[N] := [0, N-1]$. We use bold notation $\boldsymbol{v}$ to denote vectors. We define $\xleftarrow{\cup} \top$ as the operation which appends $\top$ to the data structure it was called upon. If the data structure is a set, then $\top$ is added to the set. If the data structure is a vector then $\top$ is appended to the end.

We write $\mathcal{A}^{\mathcal{B}}$ to denote that algorithm $\mathcal{A}$ has oracle access to algorithm $\mathcal{B}$ during its execution. To make the randomness ω of an algorithm $\mathcal{A}$ on input x explicit, we write $\mathcal{A}(x;\omega)$. Note that in this notation, $\mathcal{A}$ is deterministic. For a randomised algorithm $\mathcal{A}$, we use the notation $y \in \mathcal{A}(x)$ to denote that y is a possible output of $\mathcal{A}$ on input x.

Basic cryptographic assumptions and definitions used in our proofs are given in Appendix A.

3 Ratcheted Key Exchange

Throughout this paper, we consider unidirectional communication, as defined in several flavors in previous works [BSJ+17, PR18b, BRV20]. Thus, messages flow from a fixed sender to a fixed receiver; there is no communication from the receiver to the sender. We now define the syntax and properties of unidirectional ratcheted key exchangeand conceptually depict the communication flow in Figure 2.

Syntax. A unidirectional ratcheted key exchange scheme RKE consists of four algorithms $\mathsf{RKE.init}$, $\mathsf{RKE.snd}$, $\mathsf{RKE.rcv}$ and $\mathsf{RKE.rr}$, where the algorithms are defined as follows.

- $(\mathrm{stS}, \mathrm{stR}) \xleftarrow{\$} \mathsf{RKE.init}$ returns a sender and receiver state.
- $(\mathrm{stS}, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad})$ on input a sender state stS and associated data ad, outputs an updated sender state stS, a ciphertext c, and a key k.
- $(\mathrm{stR}, k) \leftarrow \mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad})$ on input a receiver state stR, a ciphertext c and associated data ad, outputs an updated receiver state stR and a key k or a failure symbol $\perp$.

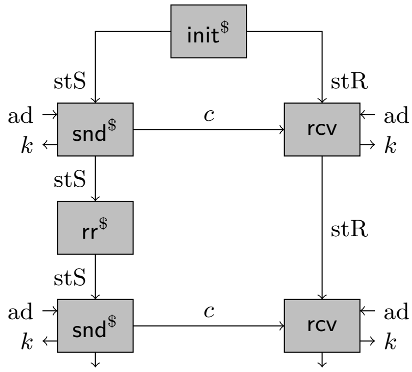

Fig. 2. Conceptual communication flow of anonymous unidirectional RKE: Alice only sends and re-randomizes her state, and Bob only receives.

– $\mathrm{stS} \xleftarrow{\$} \mathsf{RKE.rr}(\mathrm{stS})$ on input a sender state stS, outputs an randomized sender state stS.

The encapsulation space $\mathcal{C}$ and the key space $\mathcal{K}$ are defined via the support of the RKE.snd algorithm. Let $\mathcal{AD} := \{0,1\}^*$ be the space of associated data.

State Randomization. All algorithms except RKE.rr are standard in the literature of RKE. This new randomization algorithm is designed for settings in which the sender wants strong anonymity. Assume Alice has at least one running RKE session in which she sends periodically. To obfuscate both the number of running RKE sessions and the number of real ciphertexts sent in each, Alice can generate "dummy" RKE sender states. Whenever Alice executes RKE.snd with one of her states, she can re-randomize all remaining states via RKE.rr. Looking ahead, our definition of anonymity requires that all sender states are indistinguishable from a freshly generated sender state, ensuring that it is hard to identify the state that was just used for sending.[9]

Basic Consistency Requirements. In Appendix B, we specify three basic consistency notions for RKE: *Robustness*, *Correctness*, and *Recover Security*. Robustness requires that whenever algorithm $(\mathrm{stR}', k) \leftarrow \mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad})$ rejects a ciphertext c and associated data ad (and outputs $k = \perp$), the output receiver state stR' must be unchanged (i.e., $\mathrm{stR} = \mathrm{stR}'$), which is crucial for ensuring strong anonymity. Correctness requires that, as long as Bob only accepts ciphertexts sent by Alice (i.e., accepts no forged messages from the attacker), keys output by Bob match those output by Alice. Finally, recover security ensures that it is hard to perform a trivial impersonation of Alice towards Bob without being detected eventually. More concretely, whenever Bob computes a key that does not match the corresponding key computed by Alice, Bob must never accept another ciphertext from Alice.

3.1 Secrecy and Authenticity

We provide compact notions of key-indistinguishability and authenticity for RKE in B.1 and B.2. In both games, the adversary can control the protocol execution via oracles `Snd`, `RR`, `Rcv` that internally run the respective algorithms. Furthermore, the adversary can expose the sender state and receiver state via oracles $\mathtt{Expose}_S$ and $\mathtt{Expose}_R$, respectively.

Secrecy. In game KIND, which models secrecy of session keys, the adversary can additionally query `ChallSnd`. This oracle internally executes algorithm RKE.snd and, depending on random challenge bit b, either outputs the computed key k (if $b = 0$) or a uniformly random key k' (if $b = 1$). To correctly guess the challenge bit b, the adversary can query all oracles with two limitations. These limitations depend on whether the receiver accepted a ciphertext (via `Rcv`) that was not sent by the sender (via `Snd` resp. `ChallSnd`). If the receiver never accepted a malicious ciphertext, we say the receiver is *in sync*. As long as the receiver is *in sync*, querying $\mathtt{Expose}_R$ is only permitted if all ciphertexts output by `ChallSnd` were given to `Rcv` in the same order. Otherwise, exposing the receiver would reveal challenges still in transit. For the same reason, querying `ChallSnd` is forbidden if the receiver was exposed while *in sync*.

[9] A corresponding randomization algorithm for the receiver state is meaningless in the *unidirectional* RKE setting since, as soon as Bob's state is exposed, he cannot hope for any security guarantees after that.

Authenticity. In game AUTH, the adversary wins when the receiver accepts a ciphertext (via `Rcv`) that was not sent by the sender (via `Snd` resp. `ChallSnd`). The only restriction is that $\texttt{Expose}_S$ must not have been queried after the last ciphertext, accepted by the receiver *in sync* (in `Rcv`), was sent (via `Snd` resp. `ChallSnd`). This condition rules out trivial impersonations.

4 Anonymous Ratcheted Key Exchange

In anonymous ratcheted key exchange, any interaction of a fixed RKE instance, consisting of a fixed sender and receiver, should be indistinguishable from an interaction of a fresh RKE instance which is sampled uniformly at random. This includes not only the indistinguishability of ciphertexts and keys, but also the internal states. We capture these core requirements for our anonymity security experiment in so-called utopian games below.

As opposed to KIND and AUTH, there are far more trivial attacks that need to be considered. We elaborate on how we model security such that we can identify and prevent trivial attacks, and give a detailed security notion for anonymity in this section. Following the approach of Rogaway and Zhang [RZ18b], we give the core of our definition (which we call *utopian games*), ignoring trivial attacks for now.

Utopian Games. The definition of our utopian games $\mathsf{U\text{-}ANON}^b$ is given in Fig. 3. Our definitions are "real-or-random"-style and games are parameterized by a bit b, where $\mathsf{U\text{-}ANON}^0$ denotes the real world execution, and in $\mathsf{U\text{-}ANON}^1$ all outputs of challenge oracles are random. At the beginning of the game, $\mathsf{U\text{-}ANON}^b_{\mathsf{RKE}}$ samples the initial sender and receiver states and provides several oracles to the adversary. As usual for RKE security, the adversary can control the message flow and obtain internal states via oracles `Snd`, `Rcv`, `RR`, $\texttt{Expose}_S$ and $\texttt{Expose}_R$.

Game $\mathsf{U\text{-}ANON}^b_{\mathsf{RKE}}(\mathcal{A})$
$(\mathrm{stS}, \mathrm{stR}) \xleftarrow{\$} \mathsf{RKE.init}$
$\mathrm{ceStR} \leftarrow \bot$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle $\texttt{Snd}(\mathrm{ad})$
$(\mathrm{stS}, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad})$
Return (c, k)

Oracle $\texttt{Rcv}(c, \mathrm{ad})$
$(\mathrm{stR}, k) \leftarrow \mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad})$
Return $[\![k \neq \bot]\!]$:

Oracle `RR`
$\mathrm{stS} \xleftarrow{\$} \mathsf{RKE.rr}(\mathrm{stS})$
Return

Oracle $\texttt{ChallSnd}(\mathrm{ad})$
If $b = 0$:
  $(\mathrm{stS}, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad})$
If $b = 1$:
  $(\mathrm{stS}', \_) \xleftarrow{\$} \mathsf{RKE.init}$
  $(\_, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}', \mathrm{ad})$
Return (c, k)

Oracle $\texttt{Expose}_S$
Return stS

Oracle $\texttt{ChallExpose}_S$
If $b = 0$:
  Return stS
$(\mathrm{stS}', \mathrm{ceStR}) \xleftarrow{\$} \mathsf{RKE.init}$
Return stS'

Oracle $\texttt{Expose}_R$
Return stR

Oracle $\texttt{ChallExpose}_R$
If $b = 0$:
  Return stR
$(\_, \mathrm{stR}') \xleftarrow{\$} \mathsf{RKE.init}$
Return stR'

Fig. 3. Utopian games $\mathsf{U\text{-}ANON}^b$ for anonymity, where $b \in \{0, 1\}$ and RKE is a ratcheted key exchange scheme.

The remaining oracles provide the adversary with some challenge depending on b. We define three different challenge oracles:

- `ChallSnd` models indistiguishability of ciphertexts and keys. It should be hard to distinguish if the ciphertexts and keys are produced by running RKE.snd on the real sender state ($\mathsf{U\text{-}ANON}^0$) or a random sender state ($\mathsf{U\text{-}ANON}^1$).
- $\texttt{ChallExpose}_S$ models indistinguishability of sender states. In $\mathsf{U\text{-}ANON}^0$ this oracle outputs the real sender state, whereas in $\mathsf{U\text{-}ANON}^1$ it outputs a random sender state. At this point, we store the corresponding receiver state in an additional variable ceStR which we require later to define trivial attacks.
- $\texttt{ChallExpose}_R$ models indistinguishability of receiver states and is defined as in $\texttt{ChallExpose}_S$, only it instead outputs the real receiver state ($\mathsf{U\text{-}ANON}^0$) or a random receiver state ($\mathsf{U\text{-}ANON}^1$).

4.1 Anonymity Definition

In this section, we show how to extend the utopian games to a full anonymity security notion for RKE (cf. Fig. 4). Since identifying trivial attacks is quite involved and needs a lot of additional book-keeping, the subsequent text aims to give an in-depth description of our game-based definition on a syntactical level. It provides the framework to prevent trivial attacks and should help the reader to understand how all the tracing logic works. Apart from that, the security game $\mathsf{ANON}^b_{\mathsf{RKE}}$ basically builds upon the logic of the corresponding utopian game $\mathsf{U\text{-}ANON}^b$. A more high-level perspective and, in particular, descriptions of the actual trivial attacks are given in the subsequent Section 4.2.

For comprehensibility, we assume that an RKE scheme, analyzed with our anonymity notion, offers recover security, correctness, as well as authenticity. It is notable that an adversary breaking authenticity also trivially breaks anonymity (cf. Appendix C).

Execution Model. Depending on the bit b, game $\mathsf{ANON}^b_{\mathsf{RKE}}$ either simulates the real world as captured in utopian game $\mathsf{U\text{-}ANON}^0_{\mathsf{RKE}}$ or the random world as captured in utopian game $\mathsf{U\text{-}ANON}^1_{\mathsf{RKE}}$ (cf. Fig. 3). In the following, we will write $\mathsf{U\text{-}ANON}_0$ and $\mathsf{U\text{-}ANON}_1$ for brevity. Hence, ANON^b runs the utopian game $\mathsf{U\text{-}ANON}_b$ as a subroutine and we allow access to all oracles. For example, we denote oracle access by $\mathsf{U\text{-}ANON}_b.\mathtt{Snd}(\mathrm{ad})$, which will run a send query in $\mathsf{U\text{-}ANON}_b$ on input ad. We also allow access to internal variables. For example, we write $\mathsf{U\text{-}ANON}_b.\mathrm{stR}$ to access the current receiver state in $\mathsf{U\text{-}ANON}_b$.

To ensure that the game ANON^b can identify trivial attacks, we also need to observe what would have happened if we had run the same sequence of queries in the other utopian game $1-b$. We will explain this in more detail in Section 4.2. First, we introduce additional book-keeping variables and describe our oracles.

Send Queries. Oracles `Snd` and `ChallSnd` take as input a string ad which it forwards to utopian game $\mathsf{U\text{-}ANON}_b$ to compute a ciphertext and key (c, k). All tuples (c, ad) are stored in a list $\boldsymbol{cad}$. Additionally, we have counters (s_0, s_1) to keep track of the number of ciphertexts sent in game $\mathsf{U\text{-}ANON}_b$ and the number of ciphertexts that would have been sent in $\mathsf{U\text{-}ANON}_{1-b}$. On a `Snd` query, we increment both counters. Since `Snd` results in updated sender states, we already store the corresponding updated receiver state in a list $\boldsymbol{stR}$ by running the $\mathsf{RKE.rcv}$ algorithm locally (line 47). Note that the first entry of $\boldsymbol{stR}$ at position 0 is set to the initial receiver state $\mathsf{U\text{-}ANON}_b.\mathrm{stR}$ when the game is initialized (line 05). We additionally store the current counter value s_0 in a set $\boldsymbol{c}$.

On a `ChallSnd` query, we only increment s_0 because the real sender state is not used in $\mathsf{U\text{-}ANON}_1$. Thus, we also only need to store the corresponding receiver state in case $b = 0$ (line 54). The value of the counter s_0 is additionally stored in the challenge set $\boldsymbol{cc}$.

Exposures and Randomizations. Oracles $\mathtt{Expose}_S$ and $\mathtt{Expose}_R$ forward queries to the utopian game and output the real sender state stS (resp. receiver state stR). Additionally, the current sender counters (s_0, s_1) are added to a set $\boldsymbol{xS}$. We use boolean flags xS resp. xR to indicate that the sender resp. receiver was exposed.

Challenge exposures are handled similarly, however now we use a list $\boldsymbol{cxS}$ to store tuples (s_0, s_1) of a query to $\mathtt{ChallExpose}_S$. Thus, we have another list $\boldsymbol{cstR}$ to additionally store the corresponding receiver state of the exposed sender state. When $b = 0$, we simply copy the state stored in $\boldsymbol{stR}$ and for $b = 1$, we store the receiver state $\mathsf{U\text{-}ANON}_1.\mathrm{ceStR}$ (belonging to the randomly chosen sender state stS_1). We use boolean flags cxS resp. cxR to register a challenge sender resp. receiver exposure.

A randomization query via `RR` will reset the sender flags to `fal`, thus modeling post-compromise anonymity on the sender's side. Note that we do not need to track the time of a receiver exposure. Once exposed, all subsequent updated states can be computed locally by the adversary.

Before describing `Rcv` behaviour, we want to highlight the importance of impersonations. We use boolean flags imp_0, imp_1 to indicate an impersonation in $\mathsf{U\text{-}ANON}_0$ or $\mathsf{U\text{-}ANON}_1$. Both are initialized to `fal` and will be set to `tru` if a sequence of queries leads to an impersonation in the corresponding utopian game. Note that sequences of queries may lead to impersonations in both, none or one utopian game(s).[10] Thus, we need track whether an impersonation would have happened. While it is easy to check the impersonation state of the simulated game $\mathsf{U\text{-}ANON}_b$, i.e., the value of imp_b, it is more involved to determine imp_{1-b}. We will explain how this can be done below.

[10] An impersonation may occur in one of the games when sender and receiver states are not updated *simultaneously*. The sequence of oracle calls `ChallSnd`, $\mathtt{Expose}_S$ with a subsequent impersonation attempt issued to `Rcv` will only impersonate $\mathsf{U\text{-}ANON}_1$, since in $\mathsf{U\text{-}ANON}_0$ the challenge ciphertext needs to be received first.

Game $\mathsf{ANON}^b_{\mathsf{RKE}}(\mathcal{A})$

$\mathsf{U\text{-}ANON}_b \leftarrow \mathsf{U\text{-}ANON}^b_{\mathsf{RKE}}$
For $d \in \{0,1\}$:
  $(s_d, r_d) \leftarrow (0,0)$
  $\mathrm{imp}_d \leftarrow \mathtt{fal}$
$(\boldsymbol{stR}, \boldsymbol{cstR}, \boldsymbol{cad}) \leftarrow ([\cdot],[\cdot],[\cdot])$
$\boldsymbol{stR}[0] \leftarrow \mathsf{U\text{-}ANON}_b.\mathrm{stR}$
$(\boldsymbol{c}, \boldsymbol{cc}, \boldsymbol{rcvd}) \leftarrow (\emptyset,\emptyset,\emptyset)$
$(\boldsymbol{xS}, \boldsymbol{cxS}) \leftarrow (\emptyset, [\cdot])$
$(\mathrm{xS}, \mathrm{cxS}, \mathrm{xR}, \mathrm{cxR}) \leftarrow (\mathtt{fal},\mathtt{fal},\mathtt{fal},\mathtt{fal})$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle $\mathtt{RR}$

$\mathsf{U\text{-}ANON}_b.\mathtt{RR}$
· $(\mathrm{xS}, \mathrm{cxS}) \leftarrow (\mathtt{fal},\mathtt{fal})$
Return

Oracle $\mathtt{Expose}_S$

▷ If $\mathrm{cxS} = \mathtt{tru}$: Require $(s_0, \_) \notin \boldsymbol{cxS}$
▷ If $\mathrm{xS} = \mathtt{tru} \wedge (s_0, s_1) \notin \boldsymbol{xS}$:
▷   Require $(\_, s_1) \notin \boldsymbol{xS}$
⋄ If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$:
⋄   Require $\mathrm{cxR} = \mathtt{fal}$
$\mathrm{stS} \leftarrow \mathsf{U\text{-}ANON}_b.\mathtt{Expose}_S$
$\boldsymbol{xS} \xleftarrow{\cup} \{(s_0, s_1)\}$
· $\mathrm{xS} \leftarrow \mathtt{tru}$
Return stS

Oracle $\mathtt{Expose}_R$

i Require unique $= \mathtt{tru}$
◁ Require $\mathrm{cxR} = \mathtt{fal}$
⋄ Require $\mathrm{imp}_0 = \mathrm{imp}_1$
If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$:
⊕   For all $\hat{s} \in \boldsymbol{cc}$ require $\hat{s} \le r_0$
⋄   Require $(r_0, \_) \notin \boldsymbol{cxS}$
$\mathrm{stR} \leftarrow \mathsf{U\text{-}ANON}_b.\mathtt{Expose}_R$
· $\mathrm{xR} \leftarrow \mathtt{tru}$
Return stR

Oracle $\mathtt{ChallExpose}_S$

▷ If $\mathrm{xS} = \mathtt{tru} \vee \mathrm{cxS} = \mathtt{tru}$:
▷   Require $(s_0, \_) \notin \boldsymbol{cxS} \wedge (s_0, \_) \notin \boldsymbol{xS}$
⋄ If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$:
⋄   Require $\mathrm{xR} = \mathrm{cxR} = \mathtt{fal}$
$\mathrm{stS}_b \leftarrow \mathsf{U\text{-}ANON}_b.\mathtt{ChallExpose}_S$
i If $b = 0$: $\boldsymbol{cstR}.\mathsf{append}(\boldsymbol{stR}[s_0])$
i If $b = 1$: $\boldsymbol{cstR}.\mathsf{append}(\mathsf{U\text{-}ANON}_1.\mathrm{ceStR})$
$\boldsymbol{cxS}.\mathsf{append}((s_0, s_1))$
· $\mathrm{cxS} \leftarrow \mathtt{tru}$
Return stS_b

Oracle $\mathtt{Snd}(\mathrm{ad})$

⊕ If $\mathrm{imp}_0 = \mathrm{imp}_1 = \mathtt{fal}$: Require $\mathrm{cxR} = \mathtt{fal}$
$(c, k) \xleftarrow{\$} \mathsf{U\text{-}ANON}_b.\mathtt{Snd}(\mathrm{ad})$
$\boldsymbol{cad}.\mathsf{append}(c, \mathrm{ad})$
$\boldsymbol{c} \xleftarrow{\cup} \{s_0\}$
$s_0 \xleftarrow{+} 1,\ s_1 \xleftarrow{+} 1$
i $(\boldsymbol{stR}[s_b], \_) \leftarrow \mathsf{RKE.rcv}(\boldsymbol{stR}[s_b - 1], c, \mathrm{ad})$
Return (c, k)

Oracle $\mathtt{ChallSnd}(\mathrm{ad})$

⊕ If $\mathrm{imp}_0 = \mathtt{fal}$: Require $\mathrm{xR} = \mathrm{cxR} = \mathtt{fal}$
$(c_b, k_b) \xleftarrow{\$} \mathsf{U\text{-}ANON}_b.\mathtt{ChallSnd}(\mathrm{ad})$
$\boldsymbol{cad}.\mathsf{append}(c_b, \mathrm{ad})$
$\boldsymbol{cc} \xleftarrow{\cup} \{s_0\}$
$s_0 \xleftarrow{+} 1$
i If $b = 0$: $(\boldsymbol{stR}[s_0], \_) \leftarrow \mathsf{RKE.rcv}(\boldsymbol{stR}[s_0 - 1], c_0, \mathrm{ad})$
Return (c_b, k_b)

Oracle $\mathtt{Rcv}(c, \mathrm{ad})$

$\mathrm{succ}_b \leftarrow \mathsf{U\text{-}ANON}_b.\mathtt{Rcv}(c, \mathrm{ad})$
If $\exists \hat{r} \ge \min(r_0, r_1)$ s.t. $(c, \mathrm{ad}) = \boldsymbol{cad}[\hat{r}]$
  If $b = 0$:
    $r_1' \leftarrow \min(\boldsymbol{c} \setminus \boldsymbol{rcvd})$
    $\mathrm{succ}_1 \leftarrow \neg\mathrm{imp}_1 \wedge [\![r_1' = \hat{r}]\!]$
    If succ_1: $\boldsymbol{rcvd} \xleftarrow{\cup} \{\hat{r}\}$
  If $b = 1$: $\mathrm{succ}_0 \leftarrow \neg\mathrm{imp}_0 \wedge [\![r_0 = \hat{r}]\!]$
  If succ_{1-b}: $r_{1-b} \xleftarrow{+} 1$
i Else: //check for impersonations
i   Let $\mathcal{S} \subseteq \boldsymbol{xS}$ s.t. $(\_, r_1) \in \boldsymbol{xS}$
i   If $|\mathcal{S}| > 1 \wedge (r_0, \_) \in \mathcal{S}$: unique $\leftarrow \mathtt{fal}$
i   For $(\hat{r}_0, \hat{r}_1) \in \mathcal{S}$
i   If $\mathsf{RKE.rcv}(\boldsymbol{stR}[\hat{r}_b], c, \mathrm{ad}) \neq (\_, \perp)$:
i $\mathrm{imp}_0 \leftarrow \mathrm{imp}_0 \vee [\![r_0 = \hat{r}_0]\!]$
i $\mathrm{imp}_1 \leftarrow \mathtt{tru}$
i   If imp_{1-b}: $r_{1-b} \xleftarrow{+} 1$
i $\mathcal{I} \leftarrow \{ i \mid \boldsymbol{cxS}[i] = (\hat{r}_0, \hat{r}_1) \text{ s.t. } \hat{r}_b = r_b \}$
i   For $i \in \mathcal{I}$:
i   If $\mathsf{RKE.rcv}(\boldsymbol{cstR}[i], c, \mathrm{ad}) \neq (\_, \perp)$:
i $\mathrm{imp}_0 \leftarrow \mathrm{imp}_0 \vee [\![r_0 = \hat{r}_0]\!]$, where $\hat{r}_0 = \boldsymbol{cxS}[i][0]$
i   If imp_{1-b}: $r_{1-b} \xleftarrow{+} 1$
If succ_b: $r_b \xleftarrow{+} 1$
Return

Oracle $\mathtt{ChallExpose}_R$

◁ Require $\mathrm{xR} = \mathrm{cxR} = \mathtt{fal}$
⋄ Require $(r_0, \_) \notin \boldsymbol{xS} \wedge (r_0, \_) \notin \boldsymbol{cxS}$
⋄ Require $\mathrm{imp}_0 = \mathtt{fal}$
⊕ If $\mathrm{imp}_1 = \mathtt{fal}$: Require $s_0 = r_0$
$\mathrm{stR}_b \leftarrow \mathsf{U\text{-}ANON}_b.\mathtt{ChallExpose}_R$
· $\mathrm{cxR} \leftarrow \mathtt{tru}$
Return stR_b

Fig. 4. Full anonymity games ANON^b for $b \in \{0,1\}$, where lines in dashed boxes disallow trivial attacks. We further distinguish between different trivial attacks (cf. Section 4.2): Lines marked with ⊕ are due to correctness relations, those marked with ▷, ◁ are due to state equality relations on sender resp. receiver side, those marked with ⋄ are due to matching state relations, and i indicates an impersonation requirement.

Receive Queries. Oracle $\mathtt{Rcv}$ advances receiver states. Since the adversary only sees ciphertexts of $\mathsf{U\text{-}ANON}_b$, we first forward the adversary's query (c, ad) to $\mathsf{U\text{-}ANON}_b$. Similarly to the counters (s_0, s_1), we use counters (r_0, r_1) to track the number of successfully received ciphertexts in games $\mathsf{U\text{-}ANON}_0$ and $\mathsf{U\text{-}ANON}_1$. For $\mathsf{U\text{-}ANON}_{1-b}$, we can determine these numbers from the sequence of queries. We introduce another book-keeping set $\boldsymbol{rcvd}$, which stores the counter values of send queries stored in $\boldsymbol{c}$ that have been successfully received in $\mathsf{U\text{-}ANON}_1$, allowing us to keep track of which tuples stored in $\boldsymbol{cad}$ have been processed by U-ANON$_1$. Now, independent of whether this ciphertext has been received successfully, we proceed in three steps.

Check for in-order-receive (lines 57-63). If the adversary intends to receive a ciphertext output by `Snd` or `ChallSnd` (which we check by comparing the query to the list $\boldsymbol{cad}$) we need to decide if this query would have been accepted in U-ANON$_{1-b}$. Let $\hat{r}$ be the index in $\boldsymbol{cad}$ such that the tuple stored in $\boldsymbol{cad}[\hat{r}]$ matches the adversary's query. If $b = 0$, we need to decide whether this query would lead to a successful receive in U-ANON$_1$. At this point, we only care about ciphertexts from `Snd` since challenge ciphertexts in U-ANON$_1$ are produced by a random state. We denote the index of the next ciphertext in $\boldsymbol{cad}$ that belongs to a send query by r'_1. Note that we can compute r'_1 using sets $\boldsymbol{c}$ and $\boldsymbol{rcvd}$. We say that U-ANON$_1$ accepts this ciphertext if $\hat{r} = r'_1$ and we will add $\hat{r}$ to $\boldsymbol{rcvd}$. If $b = 1$, it is easy to decide whether a ciphertext would have been accepted in U-ANON$_0$, since we only need to compare $\hat{r}$ with r_0. Since any ciphertext stored in $\boldsymbol{cad}$ should not be accepted after an impersonation, the statements in lines 60, 62 will always evaluate to false.

Check for impersonations after $\texttt{Expose}_S$ (lines 65-71). We know that an exposed sender state can lead to an impersonation, depending on when exposure occurred and which ciphertexts have been received. Since we require authenticity, an impersonation can *only* occur after an exposed sender state. Thus, in U-ANON$_1$ an impersonation will only be successful if the counter value r_1 is in the set $\boldsymbol{xS}$. We add all the relevant tuples to a set $\mathcal{S}$. Ignore line 66 for now. We iterate over all entries $(\hat{r}_0, \hat{r}_1) \in \mathcal{S}$ and use $\boldsymbol{stR}[\hat{r}_b]$ to check if the ciphertext decrypts under that state. If so, this may be an impersonation, which we will decide next. Since we always have $\hat{r}_1 = r_1$, a successful decryption implies an impersonation in U-ANON$_1$, so we set imp_1 to `tru`. If $\hat{r}_0 = r_0$, then we had an impersonation in U-ANON$_0$ as well. By RECOV security, once a sender is impersonated, the receiver will no longer accept their ciphertexts. Thus once $\mathrm{imp}_0 \leftarrow \texttt{tru}$, imp_0 will always be `tru` independent of the counter comparison, which is captured by the "or" statement in line 69. The result of this check will be the same in both games ANON^0 and ANON^1, unless the case in line 66 happens. For an example of a sequence of queries triggering this case, we refer to Appendix C. Note that if there exist multiple entries such that $(\hat{r}_0, \hat{r}_1)$ in $\mathcal{S}$, but $r_0 \neq \hat{r}_0$ for all, then imp_0 will always be set to the same value.

Check for impersonations after $\texttt{ChallExpose}_S$ (lines 72-76). Impersonation can also occur using the sender state output by $\texttt{ChallExpose}_S$. Similarly to the previous step, we first identify relevant entries in the list $\boldsymbol{cxS}$. In particular, we look for all entries $(\hat{r}_0, \hat{r}_1)$, where $\hat{r}_b = r_b$. Since $\boldsymbol{cxS}$ is a list and we stored the corresponding receiver states at the same position in list $\boldsymbol{cstR}$, we need to find the position of the tuples $(\hat{r}_0, \hat{r}_1)$ and store these indices in a set $\mathcal{I}$. This structure is needed, since entries in $\boldsymbol{cxS}$ are not necessarily unique.[11] Now we proceed as in the previous step. An impersonation in U-ANON$_0$ has occurred if the counter $\hat{r}_0$ in $\boldsymbol{cxS}$ equals the current counter r_0. Note that in U-ANON$_1$, there will not be an impersonation since the real receiver state should accept a ciphertext output by a random sender state. Again, the outcome is the same for both games ANON^b. For $b = 0$, this can be observed by the fact that $\mathcal{I}$ maps to indices where $\hat{r}_0 = r_0$ and thus $\boldsymbol{cstR}[i] = \boldsymbol{cstR}[j]$ for all $i, j \in \mathcal{I}$ and the check only depends on the successful decryption using the current state. For $b = 1$, since all entries in $\boldsymbol{cstR}$ contain different receiver states, there will be at most one state that decrypts the ciphertext. Thus, $\hat{r}_0$ is uniquely defined and imp_0 is only set to `tru` if $\hat{r}_0 = r_0$ (or if it has already been `tru` before).
We will increase the counter r_{1-b} if the impersonation was successful. At the very end, we will also increase counter r_b if the query was accepted in the first place. This concludes the description of `Rcv`.

4.2 Identifying Trivial Attacks

If we ignore trivial attacks, the adversary easily distinguishes ANON^0 from ANON^1, since relations between outputs differ between games. We group these relations into four categories: ability to decrypt, state equality, matching states, and impersonations. In our pseudocode, we indicate restrictions on the adversary with a symbol corresponding to a relation group. We briefly explain the relations below, and we provide justification for all requirements in Appendix C.

[11] Imagine a sequence of queries $\texttt{ChallExpose}_S$, `RR`, $\texttt{ChallExpose}_S$. In this case, the sender counters s_0, s_1 do not change. Also the receiver states appended to $\boldsymbol{cstR}_0$ are the same, but the (random) receiver states appended to $\boldsymbol{cstR}_1$ are different, which is crucial for identifying impersonations.

Ability to Decrypt (marked with $\oplus$*).* Our correctness definition captures that a ciphertext computed with the sender state can always be decrypted with the corresponding receiver state. Due to this, lines marked with ($\oplus$) trace sequences of oracle queries that allow an adversary to determine if a given ciphertext decrypts successfully under an exposed receiver state in one game but not the other, revealing the bit b.

Equality of States (marked with $\triangleright$ *,* $\triangleleft$ *).* For both sender ($\triangleright$) and receiver ($\triangleleft$) exposures, our anonymity game allows the *direct* exposure of a real state and *challenge* exposures which will output either a real or random state. Depending on the sequence of queries, the output of two *subsequent* calls to $\texttt{Expose}_S$ or $\texttt{ChallExpose}_S$ may inevitably be the same in ANON^0 but not in ANON^1, which we detect with the marked code lines to prevent that this inconsistency trivially reveals bit b.

Matching States (marked with $\diamond$ *).* We also consider sequences of queries that may expose one party and challenge-expose the other. It is easy to see that the adversary can test whether two such states are linked (which leaks bit b) by creating a ciphertext with the exposed sender state and trial-decrypt with the receiver state.

Impersonations (marked with i *).* As argued earlier, it is crucial to determine whether a sequence of queries leads to an impersonation in any of the games ANON^0 and ANON^1. Only then, we can detect whether the relations above lead to a trivial attack. However, sometimes it is not possible to uniquely determine the impersonation status in game ANON^{1-b}. Whenever this is the case, we need to disallow receiver exposures since the receiver's state leaks whether the impersonation attempt was successful.

Finally, we formalise the advantage of an adversary against RKE anonymity.

Definition 1. *Consider the games* ANON^b *for* $b \in \{0,1\}$ *in Fig. 4. We define the advantage of an adversary* $\mathcal{A}$ *against anonymity of a ratcheted key exchange scheme* RKE *as*

$$\mathsf{Adv}^{\mathsf{ANON}}_{\mathcal{A},\mathsf{RKE}} := \left|\Pr[\mathsf{ANON}^0_{\mathsf{RKE}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{ANON}^1_{\mathsf{RKE}}(\mathcal{A}) \Rightarrow 1]\right| \ .$$

5 Updatable and Randomizable PKE

We construct two types of PKE with related properties: a randomizable PKE scheme (rPKE) and an updatable and randomizable PKE scheme (urPKE). An rPKE scheme is used in the updatable and randomizable signature scheme (cf. Section 6.2) and urPKE is a direct building block in the overall construction of ratcheted key exchange (cf. Section 7).

5.1 Randomizable PKE

In the following, we define the syntax and properties of an rPKE scheme.

Syntax. A randomizable public-key encryption scheme rPKE consists of four algorithms $\mathsf{rPKE.gen}$, $\mathsf{rPKE.enc}$, $\mathsf{rPKE.dec}$, $\mathsf{rPKE.rr}$, which are defined as follows:

- $(\mathrm{ek}, \mathrm{dk}) \xleftarrow{\$} \mathsf{rPKE.gen}$ outputs an encryption key and a decryption key.
- $c \xleftarrow{\$} \mathsf{rPKE.enc}(\mathrm{ek}, m)$ takes an ek, message m and returns an encryption c.
- $m \leftarrow \mathsf{rPKE.dec}(\mathrm{dk}, c)$ takes dk, c and outputs the decrypted message m.
- $(\mathrm{ek}, c) \xleftarrow{\$} \mathsf{rPKE.rr}(\mathrm{ek}, c)$ returns randomized ek and c.

Compared to a standard public-key encryption scheme, the additional feature lies in the $\mathsf{rPKE.rr}$ algorithm that allows to (re-)randomize encryption keys and ciphertexts while preserving correctness. More formally, we require that for all $(\mathrm{ek}, \mathrm{dk}) \in \mathsf{rPKE.gen}$, $m \in \mathcal{M}$, for random $c \xleftarrow{\$} \mathsf{rPKE.enc}(\mathrm{ek}, m)$ and for an arbitrary number of randomizations $(\mathrm{ek}, c) \xleftarrow{\$} \mathsf{rPKE.rr}(\mathrm{ek}, c)$, we have that $\mathsf{rPKE.dec}(\mathrm{dk}, c) = m$.

We want to use an rPKE scheme as building block of the signature scheme in Section 6. For this, we will need some additional properties that we define below.

Homomorphic Property. An rPKE scheme is called *homomorphic* if for an arbitrary but fixed public key $(\text{ek}, \_) \in \mathsf{rPKE.gen}$, there exists a group homomorphism $\mathsf{rPKE.enc}\colon (\mathcal{M}, \otimes) \times (\mathcal{R}, \oplus) \mapsto (\mathcal{C}, \otimes)$, where $\mathcal{M}$, $\mathcal{R}$, $\mathcal{C}$ are message space, randomness space and ciphertext space of the rPKE and $\oplus, \otimes$ are the corresponding group operations. More explicitly,

$$\mathsf{rPKE.enc}(\text{ek}, m_1; r_1) \otimes \mathsf{rPKE.enc}(\text{ek}, m_2; r_2) = \mathsf{rPKE.enc}(\text{ek}, m_1 \otimes m_2; r_1 \oplus r_2)\ ,$$

where $r_1, r_2 \in \mathcal{R}$ and $\otimes$ is taken component-wise.

Further, we want randomizations to be (computationally) indistinguishable, which we capture in the following definition.

Definition 2 (IND-R)**.** *Let* rPKE *be a randomizable public key encryption scheme. We require that a pair of encryption key and ciphertext that has been randomized via* rPKE.rr *is indistinguishable from a freshly generated encryption key and ciphertext. More formally, we define the advantage of a distinguisher $\mathcal{D}$ for arbitrary $2\ell \in \mathbb{Z}_p$, $(m_0, \ldots, m_{2\ell}) \in \mathcal{M}^{2\ell}$ as*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{D},\mathsf{rPKE}} :=\big| &\Pr[\mathcal{D}(\text{ek}, c_0, \ldots, c_\ell, \text{ek}', c'_0, \ldots, c'_\ell) \Rightarrow 1] \\ &- \Pr[\mathcal{D}(\text{ek}, c_0, \ldots, c_\ell, \hat{\text{ek}}, \hat{c}_0, \ldots, \hat{c}_\ell) \Rightarrow 1]\big|\ ,\end{aligned}$$

where $(\text{ek}, \_) \xleftarrow{\$} \mathsf{rPKE.gen}$, $c_i \xleftarrow{\$} \mathsf{rPKE.enc}(\text{ek}, m_i)$, $(\text{ek}', c'_0, \ldots, c'_\ell) \leftarrow \mathsf{rPKE.rr}(\text{ek}, c_0, \ldots, c_\ell)$, $(\hat{\text{ek}}, \_) \xleftarrow{\$} \mathsf{rPKE.gen}$, $\hat{c}_0, \ldots, \hat{c}_\ell \xleftarrow{\$} \mathsf{rPKE.enc}(\text{ek}, m_{\ell+1}, \ldots, m_{2\ell})$.

Construction. In Fig. 5, we construct an rPKE scheme based on the ElGamal KEM and PKE scheme. Thus, we denote the corresponding scheme by $\mathsf{rPKE_{EG}}$. An encryption key basically consists of an ElGamal encapsulation and KEM key. The encryption and randomization algorithms then use the homomorphic property of ElGamal.

Proc rPKE.gen	**Proc** rPKE.enc(ek, m)	**Proc** rPKE.dec(dk, c)	**Proc** rPKE.rr(ek, $c_0, \ldots, c_\ell$)
$x, r \xleftarrow{\$} \mathbb{Z}_p$	Parse ek as $(\text{ek}_0, \text{ek}_1)$	Parse c as (c_0, c_1)	Parse ek as $(\text{ek}_0, \text{ek}_1)$
$\text{dk} \leftarrow x$	$s \xleftarrow{\$} \mathbb{Z}_p$	$m \leftarrow c_1 \cdot c_0^{-\text{dk}}$	$r' \xleftarrow{\$} \mathbb{Z}_p$
$\text{ek} \leftarrow (g^r, g^{xr})$	$c_0 \leftarrow \text{ek}_0^s$	Return m	$\text{ek}' \leftarrow (\text{ek}_0^{r'}, \text{ek}_1^{r'})$
Return (ek, dk)	$c_1 \leftarrow \text{ek}_1^s \cdot m$		For $i \in [\ell]$:
	Return (c_0, c_1)		Parse c_i as (c_i^0, c_i^1)
			$s'_i \xleftarrow{\$} \mathbb{Z}_p$
			$c'_i \leftarrow (c_i^0 \cdot \text{ek}_0^{s'_i}, c_i^1 \cdot \text{ek}_1^{s'_i})$
			Return $(\text{ek}', c'_0, \ldots, c'_\ell)$

Fig. 5. Randomizable PKE scheme $\mathsf{rPKE_{EG}}$.

Lemma 1. *Scheme* $\mathsf{rPKE_{EG}}$ *is homomorphic. Furthermore, it satisfies indistinguishability of randomizations under the* DDH *assumption. In particular, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ against* DDH *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{A},\mathsf{rPKE_{EG}}} \leq \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B},\mathbb{G}}\ .$$

The proof of this lemma is straight-forward and is given in Appendix D.1.

5.2 Updatable and Randomizable PKE

In this section, we introduce the primitive of an updatable and randomizable PKE, which will be used in our construction of ratcheted key exchange. The syntax is similar to that of rPKE, but it extends it with the ability to update the key pair. We briefly sketch the differences below.

Syntax. An updatable and randomizable public-key encryption scheme urPKE consists of six algorithms urPKE.gen, urPKE.enc, urPKE.dec, urPKE.rr, urPKE.nextDk and urPKE.nextEk, where the first three algorithms are defined as for rPKE and the remaining ones follow the syntax:

– $\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$ outputs a randomized encryption key ek.
– $\mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r)$ updates the decryption key with randomness r.
– $\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r)$ updates the encryption key with randomness r.

Note that the main difference to rPKE is that the randomization algorithm urPKE.rr randomizes only the encryption key.

We now require the following additional properties.

Instance Independence. We say a urPKE scheme is *instance-independent* if for uniformly chosen randomness r and any key pair $(\mathrm{ek}, \mathrm{dk})$ in the support of urPKE.gen, the two distributions $(\mathsf{urPKE.nextEk}(\mathrm{ek}, r), \mathsf{urPKE.nextDk}(\mathrm{dk}, r))$ and $(\mathrm{ek}', \mathrm{dk}') \xleftarrow{\$} \mathsf{urPKE.gen}$ are the same.

Indistinguishability of Randomizations. Similar to rPKE, we require for IND-R (formally defined in Definition 16 in Appendix E) security that an encryption key that has been randomized is indistinguishable from a freshly generated encryption key. In particular, the two distributions $(\mathrm{ek}, \mathrm{ek}_1)$ and $(\mathrm{ek}, \mathrm{ek}_2)$, where $(\mathrm{ek}, \_) \xleftarrow{\$} \mathsf{urPKE.gen}$, $\mathrm{ek}_1 \leftarrow \mathsf{urPKE.rr}(\mathrm{ek})$, $(\mathrm{ek}_2, \_) \xleftarrow{\$} \mathsf{urPKE.gen}$ should be (computationally) indistinguishable under chosen ciphertext attacks.

Ciphertext Anonymity. For *ciphertext anonymity* of urPKE we require that ciphertexts generated by a particular (and possibly exposed) encryption key are indistinguishable from ciphertexts generated by a freshly chosen encryption key under chosen ciphertext attacks. We provide a more fine-grained game-based definition in Definition 14 in Appendix E.

Construction. We construct an updatable and randomizable PKE scheme based on hashed ElGamal, which was first proven to be IND-C secure in [ABR01]. The construction is also similar to the secretly key-updatable encryption scheme of [JMM19a], thus we will only sketch it here. We give the full scheme in Fig. 14 in Appendix D.3, including the proofs of the properties mentioned above.

Algorithms urPKE.gen, urPKE.enc, urPKE.dec follow the ideas from rPKE, only that they hash the ElGamal KEM key used for encryption. Since the ciphertext does not need to be randomized, urPKE.rr can still be performed in the same way as the randomization of the encryption key in rPKE.rr. Algorithms urPKE.nextDk and urPKE.nextEk asynchronously update the decryption and encryption key by exponentiation with some uniformly chosen randomness.

6 Updatable and Randomizable One-Time Signatures

In this section we introduce our new signature primitive, namely updatable and randomizable one-time signatures. The property of updatability refers to asynchronous updates of the signing and verification keys. Randomizability refers to the randomization of signing keys. These will be crucial to provide anonymity guarantees of our ratcheted key exchange scheme.

Challenges. The main technical difficulty in designing the signature scheme lies in maintaining unforgeability while achieving randomizability of signing keys. More specifically, randomization must be implemented in a way such that both the original signing key and one of its randomized versions produce signatures that are *unforgeable* (if neither of both signing keys is corrupted); furthermore, signatures from both signing keys must verify under the same single verification key. Simultaneously, seeing the original and the randomized signing key should be indistinguishable from seeing two independently sampled signing keys. (Note that, by unforgeability, two independent signing keys will *not* produce signatures valid under the same verification key.)

We conjecture that updatability of a signature scheme is easy for most algebraic signature schemes. Unforgeability usually reduces to hardness of inverting some one-way function mapping from signing keys to verification keys. So it must be hard to invert verification keys to get valid signing keys. Our randomization requirements, intuitively, demand this for the opposite direction, too: obtaining verification keys from signing keys must be hard. Strictly speaking, we require an even stronger property: Without having the verification key, signing keys and their signatures look random, independent of whether they correspond to the same verification key. This might seem contradictory or, at least, very strong.

Outline. As a warm-up, we start with a definition and construction of updatable one-time signatures in Section 6.1. Then, we will extend the construction to updatable and randomizable one-time signatures in

Section 6.2. To achieve randomizability, we use the ElGamal-based rPKE scheme introduced in Section 5. For a schematic overview see also Fig. 1 in the introduction.

6.1 Warm-Up: Updatable Signatures

Syntax. An updatable signature scheme uSIG consists of five algorithms $\mathsf{uSIG.gen}$, $\mathsf{uSIG.sig}$, $\mathsf{uSIG.vfy}$, $\mathsf{uSIG.nextSk}$, $\mathsf{uSIG.nextVk}$. Let $\mathcal{M}$ be the message space and $\mathcal{R}$ be the randomness space. Then the algorithms are defined as follows:

- $(\mathrm{vk}, \mathrm{sk}) \xleftarrow{\$} \mathsf{uSIG.gen}$ generates a verification key vk and signing key sk.
- $\sigma \xleftarrow{\$} \mathsf{uSIG.sig}(\mathrm{sk}, m)$ takes sk and a message m and returns a signature σ.
- $\{0,1\} \leftarrow \mathsf{uSIG.vfy}(\mathrm{vk}, m, \sigma)$ takes vk, m and σ and returns a bit indicating whether σ is a valid signature for m.
- $\mathrm{sk} \leftarrow \mathsf{uSIG.nextSk}(\mathrm{sk}, r)$ asynchronously updates sk with randomness r.
- $\mathrm{vk} \leftarrow \mathsf{uSIG.nextVk}(\mathrm{vk}, r)$ asynchronously updates vk with randomness r.

Correctness. Apart from the standard correctness requirement, we require that updates yield valid verification and signing keys. More formally, we require the following:

(1) $\forall (\mathrm{sk}, \mathrm{vk}) \in \mathsf{uSIG.gen}, m \in \mathcal{M}$:

$$\Pr[\mathsf{uSIG.vfy}(\mathrm{vk}, \sigma, m) = 1 \mid \sigma \xleftarrow{\$} \mathsf{uSIG}(\mathrm{sk}, m)] = 1$$

(2) $\forall (\mathrm{sk}, \mathrm{vk}) \in \mathsf{uSIG.gen}, r \in \mathcal{R}$:

$$(\mathsf{uSIG.nextSk}(\mathrm{sk}, r), \mathsf{uSIG.nextVk}(\mathrm{vk}, r)) \in \mathsf{uSIG.gen}$$

Intuition Updatability. At the core of our construction lies a slight variation of Lamport one time signature scheme, where signing keys are group elements. To shrink the size of signatures and to mitigate the lack of updateability we instantiate the hash function with a hash function fulfilling one-wayness and the homomorphic property. By one-wayness the unforgeability property of Lamport signature scheme is unchanged and by the homomorphic property we can i) optimize the signature length to a single element in the target group ii) update signing and verification key.

To achieve this we use pairings. Let $\mathcal{G}$ be a pairing group with bilinear map $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. By the XDH assumption, DDH is hard in group $\mathbb{G}_1$ and CDH is hard in groups $\mathbb{G}_1$ and $\mathbb{G}_2$. For fixed $g_2 \in \mathbb{G}_2$, we then set $\mathsf{H}(h) := e(h, g_2)$. Clearly the homomorphic property of H follows from bilinearity of the pairing,

$$e(m_1, g_2) \cdot e(m_2, g_2) = e(m_1 \cdot m_2, g_2) .$$

By the FAPI-2 Assumption [GHV07], H is a one way function.

CONSTRUCTION. Our construction of an updatable one-time signature scheme is given in Fig. 6. It follows the idea of the one-time Lamport signature scheme, where we replace the hash function of the original scheme with a Type-II pairing. Thus, let $\mathcal{G}$ be a pairing group (cf. Definition 7) and $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\ell$ a hash function. Secret keys consist of 2ℓ group elements in $\mathbb{G}_1$ and verification keys consist of 2ℓ group elements in $\mathbb{G}_T$. For the signature generation, we borrow the approach of aggregated BLS signatures [BLS01, BGLS03]. Additionally following the "Hash-and-Sign" approach, we first hash the message using H and then interpret the hash value bit-wise. For the ith bit we choose the ith element of the signing key depending on the bit value. The signature σ will then be the product of ℓ group elements. Verification uses the pairing to compute $e(\sigma, g_2)$ and compares the result to the product of the respective ℓ target group elements.

The idea for updating the signing and verification key is that we can multiply each group element of the signing key $\mathrm{sk}_{i,b}$ with another group element $R_{i,b}$. Verification keys can be updated by multiplying the respective target group element with $e(R_{i,b}, g_2)$.

In Appendix E.2 we prove one-time existential unforgeability of the scheme (cf. Definition 15).

6.2 Extension to Updatable and Randomizable Signatures

Syntax. An updatable and randomizable signature scheme urSIG shares the syntax of an updatable signature scheme, i.e., the algorithms $\mathsf{urSIG.gen}$, $\mathsf{urSIG.sig}$, $\mathsf{urSIG.vfy}$, $\mathsf{urSIG.nextSk}$, $\mathsf{urSIG.nextVk}$ are defined analogously. Additionally, there is a sixth algorithm $\mathsf{urSIG.rr}$, which is defined as follows

- $\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$ randomizes the signing key sk.

Proc $\mathsf{uSIG.gen}$

For $b \in \{0,1\}, i \in [\ell]$:

  $x_{i,b} \xleftarrow{\$} \mathbb{Z}_p$

$\mathrm{sk} \leftarrow \begin{pmatrix} g_1^{x_{0,0}}, \ldots, g_1^{x_{\ell-1,0}} \\ g_1^{x_{0,1}}, \ldots, g_1^{x_{\ell-1,1}} \end{pmatrix}$

$\mathrm{vk} \leftarrow \begin{pmatrix} e(g_1^{x_{0,0}}, g_2), \ldots, e(g_1^{x_{\ell-1,0}}, g_2) \\ e(g_1^{x_{0,1}}, g_2), \ldots, e(g_1^{x_{\ell-1,1}}, g_2) \end{pmatrix}$

Return $(\mathrm{vk}, \mathrm{sk})$

Proc $\mathsf{uSIG.nextSk}(\mathrm{sk}, R \in \mathbb{G}_1^{2\times\ell})$

For $b \in \{0,1\}, i \in [\ell]$:

  $\mathrm{sk}_{i,b} \leftarrow \mathrm{sk}_{i,b} \cdot R_{i,b}$

Return sk

Proc $\mathsf{uSIG.sig}(\mathrm{sk}, m)$

Parse $(h_0, \ldots, h_{\ell-1}) \leftarrow \mathsf{H}(m)$ as bits

$\sigma \leftarrow \prod_{i\in[\ell]} \mathrm{sk}_{i,h_i} = g_1^{\sum x_{i,h_i}}$

Return $\sigma \in \mathbb{G}_1$

Proc $\mathsf{uSIG.vfy}(\mathrm{vk}, m, \sigma)$

Parse $(h_0, \ldots, h_{\ell-1}) \leftarrow \mathsf{H}(m)$ as bits

Return $e(\sigma, g_2) = \prod_{i=0}^{\ell-1} \mathrm{vk}_{i,h_i} = e(g_1^{\sum x_{i,h_i}}, g_2)$

Proc $\mathsf{uSIG.nextVk}(\mathrm{vk}, R \in \mathbb{G}_1^{2\times\ell})$

For $b \in \{0,1\}, i \in [\ell]$:

  $\mathrm{vk}_{i,b} \leftarrow \mathrm{vk}_{i,b} \cdot e(R_{i,b}, g_2)$

Return vk

Fig. 6. Updatable one-time signature scheme uSIG for a pairing group $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where $\mathsf{H}\colon \{0,1\}^* \mapsto \{0,1\}^\ell$ is a hash function.

Correctness. We extend correctness requirements (1), (2) from the previous section by the following: We require that for all $(\mathrm{vk}, \mathrm{sk}) \in \mathsf{urSIG.gen}$, $m \in \mathcal{M}$, an arbitrary number of randomizations resulting in an randomized signing key $\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$, a signature $\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, m)$ still verifies correctly.

Below we define a similar security property as for randomizable PKE schemes, which will be needed in the anonymity proof of our ratcheted key exchange scheme.

We define additional security properties that are needed for authenticity and recover security in Appendix E.1.

Definition 3 (Indistinguishability of Randomizations). *Let* urSIG *be a an updatable and randomizable signature scheme. We require that a signing key that has been randomized using* $\mathsf{urSIG.rr}$ *is indistinguishable from a freshly generated signing key. More formally, we define the advantage of a distinguisher* $\mathcal{D}$ *as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{D},\mathsf{urSIG}} := |\Pr[\mathcal{D}(\mathrm{sk}, \mathrm{sk}_0) \Rightarrow 1] - \Pr[\mathcal{D}(\mathrm{sk}, \mathrm{sk}_1) \Rightarrow 1]| \ ,$$

where the probability is taken over $(\mathrm{sk}, \mathrm{vk}) \xleftarrow{\$} \mathsf{urSIG.gen}$, $\mathrm{sk}_0 \leftarrow \mathsf{urSIG.rr}(\mathrm{sk})$ *and* $(\mathrm{sk}_1, \_) \xleftarrow{\$} \mathsf{urSIG.gen}$ *and the internal randomness of* $\mathcal{D}$.

Our Construction. In Fig. 7 we extend the updatable signature scheme in Fig. 6 by the randomizable PKE in Fig. 5 to get an updatable and randomizable one-time signature scheme.

Recall that signing keys in our updatable one-time signature scheme are group elements. In order to achieve signing key randomization, the idea is to encrypt those signing keys with ElGamal. However, this means that the ElGamal encryption key must be part of the overall signing key and thus in turn be randomized as well. Therefore, we do not use plain ElGamal encryption, but our randomizable PKE encryption scheme $\mathsf{rPKE}_{\mathsf{EG}}$.

Finally, to achieve strong unforgeability we use the CHK transformation [MRY04, CHK04] using a strongly unforgeable signature.

7 Construction of Anonymous RKE

Our construction of anonymous unidirectional RKE in Figure 8 elegantly arises from the two primitives presented in the last sections, urPKE and urSIG. Beyond this, we use a hash function (modeled as a random oracle) and a pseudorandom generator (PRG).

Construction. On initialization, a urPKE key pair and a urSIG key pair is generated, both of which are split between Alice's and Bob's state. Randomization of Alice's state works componentwise. When sending, Alice (1) generates a fresh signature key pair, (2) encrypts the new verification key as well as random symmetric keys, and (3) signs the resulting ciphertext with her prior signing key. (4) The signature is encrypted with one of the encrypted symmetric keys. Using the random oracle on input of the other symmetric key, the composed ciphertext, and the associated data string, Alice (5) derives the

Proc $\mathsf{urSIG.gen}$

00 $(\mathrm{ek}, \mathrm{dk}) \leftarrow \mathsf{rPKE.gen}$
01 $(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{uSIG.gen}$
02 For $b \in \{0,1\}, i \in [\ell]$:
03 $\quad (\mathrm{sk}^{(r)}_{i,b}, \mathrm{sk}^{(x)}_{i,b}) \leftarrow \mathsf{rPKE.enc}(\mathrm{ek}, m = \mathrm{sk}'_{i,b})$
04 $\mathrm{sk}^{(r)} \leftarrow \begin{pmatrix} \mathrm{sk}^{(r)}_{0,0}, \ldots, \mathrm{sk}^{(r)}_{\ell-1,0} \\ \mathrm{sk}^{(r)}_{0,1}, \ldots, \mathrm{sk}^{(r)}_{\ell-1,1} \end{pmatrix} = \begin{pmatrix} g^{r_{0,0}}, \ldots, g^{r_{\ell-1,0}} \\ g^{r_{0,1}}, \ldots, g^{r_{\ell-1,1}} \end{pmatrix}$
05 $\mathrm{sk}^{(x)} \leftarrow \begin{pmatrix} \mathrm{sk}^{(x)}_{0,0}, \ldots, \mathrm{sk}^{(x)}_{\ell-1,0} \\ \mathrm{sk}^{(x)}_{0,1}, \ldots, \mathrm{sk}^{(x)}_{\ell-1,1} \end{pmatrix} = \begin{pmatrix} g^{r_{0,0}\mathrm{dk}} g^{x_{0,0}}, \ldots, g^{r_{\ell-1,0}\mathrm{dk}} g^{x_{\ell-1,0}} \\ g^{r_{0,1}\mathrm{dk}} g^{x_{0,1}}, \ldots, g^{r_{\ell-1,1}\mathrm{dk}} g^{x_{\ell-1,1}} \end{pmatrix}$
06 $\mathrm{vk} \leftarrow (\mathrm{vk}', \mathrm{dk}); \mathrm{sk} \leftarrow (\mathrm{ek}, (\mathrm{sk}^{(r)}_{i,b}, \mathrm{sk}^{(x)}_{i,b}))$
07 Return $(\mathrm{vk}, \mathrm{sk})$

Proc $\mathsf{urSIG.sig}(\mathrm{sk}, m)$

08 $\sigma_r \leftarrow \mathsf{uSIG.sig}(\mathrm{sk}^{(r)}, m) = \prod g^{r_{i,h_i}}$
09 $\sigma_x \leftarrow \mathsf{uSIG.sig}(\mathrm{sk}^{(x)}, m) = \prod g^{r_{i,h_i}\mathrm{dk}} g^{x_{i,h_i}}$
10 Return (σ_r, σ_x)

Proc $\mathsf{urSIG.rr}(\mathrm{sk})$

11 Return $\mathsf{rPKE.rr}(\mathrm{sk})$

Proc $\mathsf{urSIG.vfy}(\mathrm{vk}, m, \sigma)$

12 Parse $(\mathrm{vk}', \mathrm{dk}) \leftarrow \mathrm{vk}$
13 $\sigma' \leftarrow \mathsf{rPKE.dec}(\mathrm{dk}, \sigma) = \prod g^{x_{i,h_i}}$
14 Return $\mathsf{uSIG.vfy}(\mathrm{vk}', \sigma', m)$

Proc $\mathsf{urSIG.nextSk}(\mathrm{sk}, r)$

15 Return $\mathsf{uSIG.nextSk}(\mathrm{sk}, r)$

Proc $\mathsf{urSIG.nextVk}(\mathrm{vk}, r)$

16 Return $\mathsf{uSIG.nextVk}(\mathrm{vk}, r)$

Fig. 7. Our updatable and randomizable one-time signature scheme $\mathsf{urSIG}[\mathsf{rPKE}, \mathsf{uSIG}]$.

Proc $\mathsf{RKE.init}$

00 $(\mathrm{ek}, \mathrm{dk}) \xleftarrow{\$} \mathsf{urPKE.gen}$
01 $(\mathrm{vk}, \mathrm{sk}) \xleftarrow{\$} \mathsf{urSIG.gen}$
02 $\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
03 $\mathrm{stR} \leftarrow (\mathrm{dk}, \mathrm{vk})$
04 Return $(\mathrm{stS}, \mathrm{stR})$

Proc $\mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad})$

05 $(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
06 $k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
07 $(\mathrm{vk}_{\mathrm{next}}, \mathrm{sk}_{\mathrm{next}}) \leftarrow \mathsf{urSIG.gen}$
08 $c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}))$
09 $\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
10 $c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
11 $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
12 $\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
13 $\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r_{\mathsf{urPKE}})$
14 $\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
15 Return (stS, k, c)

Proc $\mathsf{RKE.rr}(\mathrm{stS})$

16 $(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
17 $\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$
18 $\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$
19 $\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
20 Return stS

Proc $\mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad})$

21 $k \leftarrow \perp$
22 $(\mathrm{dk}, \mathrm{vk}) \leftarrow \mathrm{stR}$
23 $(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
24 $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
25 Require $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \perp$
26 If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
27 $\quad (k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
28 $\quad \mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
29 $\quad \mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r_{\mathsf{urPKE}})$
30 $\quad \mathrm{stR} \leftarrow (\mathrm{dk}, \mathrm{vk})$
31 Return (stR, k)

Fig. 8. Construction of our RKE scheme $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$.

final session key as well as two pseudorandom strings which update her two state components (encryption key and signing key). Bob performs the corresponding decryption, verification, hash evaluation, and key updates when receiving.

Consistency and Authenticity. By the *correctness* properties of urPKE and urSIG, this URKE construction is correct, too. The construction provides *robustness* since Bob either accepts with an actual session key (if decryption and verification succeed) or his state remains unchanged. We formally prove *recover security* of this construction in Appendix F.2. On an intuitive level, each fresh signing key is "entangled" with the ciphertext that transmits it via the key update in line 12. This means that Bob will only accept signatures from a signing key if he received the corresponding verification key with the originally transmitted ciphertext. Based on unforgeability of the urSIG scheme and collision resistance of the random oracle, this mechanism maintains recover security. *Authenticity* similarly follows from the signature scheme's unforgeability, which we prove in Appendix F.3.

Secrecy. In the presence of a passive adversary, the secrecy of session keys follows directly from the confidentiality of the urPKE scheme. In case of a trivial impersonation—which, by authenticity, is the only successful way to let Bob accept a forged ciphertext—, we need the consistency guarantees of the urSIG scheme and the hash function to prove that Bob's state immediately diverges incompatibly from Alice's state. We prove this informal claim in Appendix F.4.

Anonymity. Below we establish our main theorem, namely anonymity of our RKE construction. Additionally, we provide theorems and proofs for robustness, recover security, authenticity and key indistinguishability in Appendices F.1 to F.4.

Theorem 1 (Anonymity of $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$). *Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a random oracle. Let* urPKE *be an updatable and randomizable PKE scheme. Let* urSIG *be an updatable and randomizable one-time signature scheme. Let* PRG *be a pseudorandom generator. We show that* $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$ *is secure with respect to* ANON*, such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{RKE}} \leq (q_S + q_{CS}) \cdot \mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{urPKE}} + q_{CS} \cdot \mathsf{Adv}_{\mathsf{PRG}} \\ + (q_{CE} + q_{CS}) \cdot (\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urSIG}} + \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urPKE}_{\mathsf{EG}}}) + \frac{1}{2^\lambda} \ .\end{aligned}$$

where q_S, q_{CS}, and q_{CE} are the number of queries to oracles `Snd`*,* `ChallSnd`*, and* $\mathtt{ChallExpose}_S$*, respectively.*

We provide a proof sketch below and defer the full proof to Appendix F.5.

Proof (Sketch). Conceptually, the proof consists of three steps. First we show on the sender side that after calls to oracles `Snd` and `ChallSnd`, the sender states are statistically independent from prior ones. Similarly, after successful calls to oracle `Rcv`, the receiver state is statistically independent from prior ones. The forward *anonymity* and post-compromise *anonymity* guarantees follow from this state independence. We prove this independence via $(q_S + q_{CS})$ applications of the instance independence of urPKE.

In the second step, we replace all outputs of challenge oracles in the real world with independently sampled values. We get this for free for oracle $\mathtt{ChallExpose}_R$, since, by definition of our trivial attack detection and instance independence, the adversary may call oracle $\mathtt{ChallExpose}_R$ only on receiver states which are statistically independent from any other oracle output. To replace the output of oracle `ChallSnd` with random, we employ two hybrid arguments. In the first hybrid argument, we show that the adversary cannot distinguish whether we replaced challenge ciphertexts c_{urPKE} with random ciphertexts, implying a loss factor of $(q_S + q_{CS}) \cdot \mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{urPKE}}$. In the second hybrid argument, we replace all outputs of the PRG in oracle `ChallSnd` with random, implying a loss factor of $q_{CS} \cdot \mathsf{Adv}_{\mathsf{PRG}}$. To replace the outputs of oracle $\mathtt{ChallExpose}_S$ with uniform random values, we again give two hybrid arguments. Here we loose a total factor of $q_{CE} \cdot (\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urSIG}} + \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urPKE}_{\mathsf{EG}}})$. Finally, in the third step of the proof, we show that the adversary cannot distinguish how often the sender state was advanced. Recall that oracle `ChallSnd` is the only oracle which updates the sender state depending on bit b. In order for the adversary to see a difference in updated sender states, the adversary must expose the sender prior to and after a call to oracle `ChallSnd`. By definition of the trivial attacks, the adversary must call oracle `RR` before exposing the sender a second time. Using a hybrid argument, we replace the sender state after a call to `RR` by uniform random values in both worlds. Thus the adversary learns with both sender state exposures two independent distributions of sender states, which implies a total loss factor of $q_{CS} \cdot (\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urSIG}} + \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urPKE}_{\mathsf{EG}}})$.

Acknowledgments We thank Kenny Paterson, Eike Kiltz, and Joël Alwen for recurring very inspiring discussions during our work on this article. Special thanks goes to Kenny for hosting Paul as a visitor at ETH Zürich, which led to launching this research project.

Doreen Riepel was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

AAN+22. Joël Alwen, Benedikt Auerbach, Miguel Cueto Noval, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. CoCoA: Concurrent continuous group key agreement. In *EUROCRYPT 2022*, LNCS, 2022.

ABR01. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001.

ACD19. Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.

ACDT20. Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.

BBDP01. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, December 2001.

BBR+22. Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-14, IETF, 2022. Work in Progress.

BBS04. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

BDG+22. Alexander Bienstock, Yevgeniy Dodis, Sanjam Garg, Garrison Grogan, Mohammad Hajiabadi, and Paul Rösler. On the worst-case inefficiency of CGKA. In *TCC 2022*, LNCS. Springer, 2022.

BDR20. Alexander Bienstock, Yevgeniy Dodis, and Paul Rösler. On the price of concurrency in group ratcheting protocols. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 198–228. Springer, Heidelberg, November 2020.

BGLS03. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

BR04. Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. https://eprint.iacr.org/2004/331.

BRV20. Fatih Balli, Paul Rösler, and Serge Vaudenay. Determining the core primitive for optimally secure ratcheting. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 621–650. Springer, Heidelberg, December 2020.

BSJ+17. Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650. Springer, Heidelberg, August 2017.

CCG+18. Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, October 2018.

CDV21. Andrea Caforio, F. Betül Durak, and Serge Vaudenay. Beyond security and efficiency: On-demand ratcheting with security awareness. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 649–677. Springer, Heidelberg, May 2021.

CHK04. Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 207–222. Springer, Heidelberg, May 2004.

DHRR22a. Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly anonymous ratcheted key exchange. In *ASIACRYPT 2022*, LNCS, 2022.

DHRR22b. Benjamin Dowling, Eduard Hauck, Doreen Riepel, and Paul Rösler. Strongly anonymous ratcheted key exchange. Cryptology ePrint Archive, 2022.

DKW21. Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. In Kobbi Nissim and Brent Waters, editors, *TCC 2021*, volume 13044 of *LNCS*, pages 254–285. Springer, 2021.

DRS20. Benjamin Dowling, Paul Rösler, and Jörg Schwenk. Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 341–373. Springer, Heidelberg, May 2020.

DS18. Jean Paul Degabriele and Martijn Stam. Untagging Tor: A formal treatment of onion encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 259–293. Springer, Heidelberg, April / May 2018.

DV19. F. Betül Durak and Serge Vaudenay. Bidirectional asynchronous ratcheted key agreement with linear complexity. In Nuttapong Attrapadung and Takeshi Yagi, editors, *IWSEC 19*, volume 11689 of *LNCS*, pages 343–362. Springer, Heidelberg, August 2019.

EKN+22. Keita Emura, Kaisei Kajita, Ryo Nojima, Kazuto Ogawa, and Go Ohtake. Membership privacy for asynchronous group messaging. Cryptology ePrint Archive, Report 2022/046, 2022. https://eprint.iacr.org/2022/046.

Fis07. Marc Fischlin. Anonymous signatures made easy. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 31–42. Springer, Heidelberg, April 2007.

GHV07. S. D. Galbraith, F. Hess, and F. Vercauteren. Aspects of pairing inversion. Cryptology ePrint Archive, Report 2007/256, 2007. https://eprint.iacr.org/2007/256.

GMP22. Paul Grubbs, Varun Maram, and Kenneth G. Paterson. Anonymous, robust post-quantum public key encryption. In *EUROCRYPT 2022*, LNCS, 2022.

HWZ07. Qiong Huang, Duncan S. Wong, and Yiming Zhao. Generic transformation to strongly unforgeable signatures. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 1–17. Springer, Heidelberg, June 2007.

IY22. Ren Ishibashi and Kazuki Yoneyama. Post-quantum anonymous one-sided authenticated key exchange without random oracles. In *PKC 2022*, page 35–65, 2022.

JMM19a. Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Heidelberg, May 2019.

JMM19b. Daniel Jost, Ueli Maurer, and Marta Mularczyk. A unified and composable take on ratcheting. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 180–210. Springer, Heidelberg, December 2019.

JS18. Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

KMO+13. Markulf Kohlweiss, Ueli Maurer, Cristina Onete, Björn Tackmann, and Daniele Venturi. Anonymity-preserving public-key encryption: A constructive approach. In Emiliano De Cristofaro and Matthew K. Wright, editors, *PETS 2013*, volume 7981 of *LNCS*, pages 19–39. Springer, Heidelberg, July 2013.

Lam79. Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

MKA+21. Ian Martiny, Gabriel Kaptchuk, Adam Aviv, Dan Roche, and Eric Wustrow. Improving signal's sealed sender. 2021.

MRY04. Philip D. MacKenzie, Michael K. Reiter, and Ke Yang. Alternatives to non-malleability: Definitions, constructions, and applications (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 171–190. Springer, Heidelberg, February 2004.

Per18. Trevor Perrin. The noise protocol framework. http://noiseprotocol.org/noise.html, 2018. Revision 34.

PM16. Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf, 11 2016.

PR18a. Bertram Poettering and Paul Rösler. Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296, 2018. https://eprint.iacr.org/2018/296.

PR18b. Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018.

RMS18. Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema. In *IEEE EuroS&P 2018*, pages 415–429. IEEE, 2018.

RZ18a. Phillip Rogaway and Yusi Zhang. Onion-ae: Foundations of nested encryption. *Proc. Priv. Enhancing Technol.*, 2018(2):85–104, 2018.

RZ18b. Phillip Rogaway and Yusi Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018.

Sig18. Signal. Sealed sender. https://signal.org/blog/sealed-sender/, 2018. Blog post.

SSL20. Sven Schäge, Jörg Schwenk, and Sebastian Lauer. Privacy-preserving authenticated key exchange and the case of IKEv2. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 567–596. Springer, Heidelberg, May 2020.

TLMR22. Nirvan Tyagi, Julia Len, Ian Miers, and Thomas Ristenpart. Orca: Blocklisting in sender-anonymous messaging. In *USENIX Security 2022*, 2022.

VGIK20. Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. On the cryptographic deniability of the Signal protocol. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20, Part II*, volume 12147 of *LNCS*, pages 188–209. Springer, Heidelberg, October 2020.

YWDW06. Guomin Yang, Duncan S. Wong, Xiaotie Deng, and Huaxiong Wang. Anonymous signature schemes. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 347–363. Springer, Heidelberg, April 2006.

Zha16. Yunlei Zhao. Identity-concealed authenticated encryption and key exchange. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1464–1479. ACM Press, October 2016.

A Omitted Preliminaries

A.1 Notation for Security Games

We use standard code-based security games [BR04]. A *game* $\mathbf{G}$ is a probability experiment in which an adversary $\mathcal{A}$ interacts with an implicit challenger that answers oracle queries issued by $\mathcal{A}$. The game $\mathbf{G}$ has one *main procedure* and an arbitrary amount of additional *oracle procedures* which describe how these oracle queries are answered. We denote the (binary) output b of game $\mathbf{G}$ between a challenger and an adversary $\mathcal{A}$ as $\mathbf{G}^{\mathcal{A}} \Rightarrow b$. $\mathcal{A}$ is said to *win* $\mathbf{G}$ if $\mathbf{G}^{\mathcal{A}} \Rightarrow 1$. Unless otherwise stated, the randomness in the probability term $\Pr[\mathbf{G}^{\mathcal{A}} \Rightarrow 1]$ is over all the random coins in game $\mathbf{G}$.

A.2 Assumptions

Definition 4 (CDH). *Let $\mathbb{G}$ be a group of prime order p with generator g. We define the advantage of an PPT adversary $\mathcal{A}$ against the computational Diffie-Hellman (CDH) problem as follows*

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{D},\mathbb{G}} := \Pr_{a,b \xleftarrow{\$} \mathbb{Z}_p}\left[\mathcal{A}(g, g^a, g^b) \Rightarrow g^{ab}\right] . \tag{1}$$

Definition 5 (DDH). *Let $\mathbb{G}$ be a group of prime order p with generator g. We define the advantage of a distinguisher $\mathcal{D}$ against the decisional Diffie-Hellman (DDH) problem as follows*

$$\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{D},\mathbb{G}} := \left| \Pr_{a,b \xleftarrow{\$} \mathbb{Z}_p}[\mathcal{D}(g, g^a, g^b, g^{ab}) \Rightarrow 1] - \Pr_{a,b,c \xleftarrow{\$} \mathbb{Z}_p}[\mathcal{D}(g, g^a, g^b, g^c) \Rightarrow 1] \right| . \tag{2}$$

We say (g^a, g^b, g^{ab}) is the real DDH *tuple and g^a, g^b, g^c, where $c \neq ab$ is the random* DDH *tuple.*

Definition 6 (q-GDH). *Let $\mathbb{G}$ be a group of prime order p with generator g. We define the advantage of an adversary $\mathcal{A}$ against the q-fold gap Diffie-Hellman (q-GDH) problem as follows*

$$\mathsf{Adv}^{q\text{-}\mathsf{GDH}}_{\mathcal{A},\mathbb{G}} := \Pr_{(a_i, b_i \xleftarrow{\$} \mathbb{Z}_p)_{i \in [q]}}\left[\mathcal{A}^{\mathtt{DDH}(\cdot)}(g, (g^{a_i}, g^{b_i})_{i \in [q]}) \Rightarrow g^{a_{i^*} b_{i^*}}\right] , \tag{3}$$

where $i^* \in [q]$ and oracle $\mathtt{DDH}(X, Y, Z)$ is a decisional Diffie-Hellmann oracle which returns a bit indicating whether (X, Y, Z) is a real or a random DDH *tuple.*

Definition 7 (Pairing Groups). *Let $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be a description of a pairing group, where $\mathbb{G}_1 \neq \mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of prime order p. $\mathbb{G}_1$ and $\mathbb{G}_2$ are generated by g_1 and g_2, respectively. $e : \mathbb{G}_1 \times \mathbb{G}_2 \Rightarrow \mathbb{G}_T$ is a non-degenerate bilinear map (also called pairing). We consider Type-II pairing groups, where there is no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$.*

Definition 8 (XDH Assumption [BBS04]). *Let $\mathcal{G}$ be a description of a pairing group as defined above. We define the advantage of a distinguisher $\mathcal{D}$ against the External Decisional Diffie-Hellmann Assumption (XDH) as*

$$\mathsf{Adv}^{\mathsf{XDH}}_{\mathcal{D},\mathcal{G}} := \left| \Pr_{a,b \xleftarrow{\$} \mathbb{Z}_p}[\mathcal{D}(g_1^a, g_1^b, g_1^{ab}) \Rightarrow 1] - \Pr_{a,b,c \xleftarrow{\$} \mathbb{Z}_p}[\mathcal{D}(g_1^a, g_1^b, g_1^c) \Rightarrow 1] \right| . \tag{4}$$

Definition 9 (FAPI-2 Assumption [GHV07]). *Let $\mathcal{G}$ be a description of a pairing group as defined above. We define the advantage of an adversary $\mathcal{A}$ against the Fixed Argument Pairing Inversion 2 Problem (FAPI-2) as*

$$\mathsf{Adv}^{\mathsf{FAPI\text{-}2}}_{\mathcal{A},\mathcal{G}} := \Pr_{a \xleftarrow{\$} \mathbb{Z}_p}\left[\mathcal{A}(g_1, g_2, e(g_1^a, g_2)) \Rightarrow g_1^a\right] . \tag{5}$$

Note that this assumption is implied by the computational Diffie-Hellman problem in group $\mathbb{G}_1$.

A.3 Primitives

Pseudorandom Generators. Pseudorandom Generators (PRG) are functions that expand truly random input to longer pseudorandom output. Let $n \in \mathbb{N}, \ell = poly(n), \ell > n, \mathsf{PRG}_n \colon \{0,1\}^n \mapsto \{0,1\}^\ell$. Let $\mathcal{A}$ be an adversary on input ℓ-bit string outputs a bit. For PRG security we require that the adversary cannot distinguish between distributions $\{0,1\}^\ell$ and $\mathsf{PRG}(\xleftarrow{\$} \{0,1\}^n)$.

Lemma 2. *If* 1-GDH *is hard then* q-GDH *is hard, where* $\mathsf{Adv}_{\mathbb{G}}^{q\text{-GDH}} \leq \mathsf{Adv}_{\mathbb{G}}^{1\text{-GDH}}$.

Proof. Let $\mathcal{A}$ be an adversary against the q-GDH assumption. We show how to construct an adversary $\mathcal{B}$ against 1-GDH. Adversary $\mathcal{B}$ is called on an instance g, g^a, g^b. To simulate, adversary $\mathcal{B}$ samples q values $(r_i)_{i\in[q]}, (s_i)_{i\in[q]}$, and computes $(g^{a_i}, g^{b_i})_{i\in[q]} \leftarrow ((g^a)^{r_i}, (g^b)^{s_i})_{i\in[q]}$. To simulate oracle DDH, adversary $\mathcal{B}$ forwards the queries to its own DDH oracle. To extract from $\mathcal{A}$'s output A, which is a solution to the 1-GDH problem, $\mathcal{B}$ queries $\mathtt{DDH}((g^a)^{r_i}, (g^b)^{s_i}, A^{\frac{1}{r_i s_i}})$ for all r_i, s_i until the oracle outputs 1. If this is the case for some i, adversary $\mathcal{B}$ returns $A^{\frac{1}{r_i s_i}}$.

B Omitted Definitions for Ratcheted Key Exchange

Robustness. Similar to [BSJ+17], we define a robustness requirement.

Definition 10 (Robustness). *We require that for every pair of states* $(\mathrm{stS}, \mathrm{stR})$ *in the support of* RKE.init, *for all adversaries* $\mathcal{A}$, *for any* $(c, \mathrm{ad}) \xleftarrow{\$} \mathcal{A}(\mathrm{stS}, \mathrm{stR})$ *and* $(\mathrm{stR}', k_R) \leftarrow \mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad})$:

- $\mathrm{stR}' \neq \bot$, *and*
- *if* $k_R = \bot$, *then* $\mathrm{stR}' = \mathrm{stR}$.

We say that RKE *is robust if it fulfills the robustness requirement.*

This models that schemes are robust to bad inputs. In particular, if the receiver rejects an input (c, ad), then the receiver's state will not change. In the following, we will only consider robust schemes.

Correctness. We define game CORREC in Fig. 9, where we give the adversary $\mathcal{A}$ full control over the message flows. That is, the game first produces a pair of states $(\mathrm{stS}, \mathrm{stR})$ using the RKE.init algorithm and then the adversary is given access to oracles Snd, RR and Rcv, which will perform the corresponding algorithm on the game states and adversarial inputs, and oracle Expose, which will output the internal sender and receiver state.

Intuitively, an RKE scheme is correct if encapsulations output by Snd that are received by Rcv using the same additional data ad and maintaining the order, produce the same keys $k_S = k_R$. To keep track of the ordering, we use an array ***cadk*** which holds a tuple of additional data ad, the encapsulation c and the key k_S for each query to Snd. As long as all outputs of Snd are delivered to Rcv in the same order, without any interference, sender and receiver are in-sync, indicated by variable is = tru. We additionally use counters s and r to keep track of the number of Snd queries and in-order Rcv queries. When Rcv is queried on an input (c, ad), the game runs the RKE.rcv algorithm to obtain a (potentially updated) receiver state stR and a key k_R. If $k_R = \bot$, the oracle directly returns. Otherwise, we check the ordering using ***cadk***. That is, we compare the entry in ***cadk*** at position r with the input to Rcv. If the order was changed, we set is = fal. If the order was maintained, we check if the receiver key k_R equals the sender key k_S stored in ***cadk***. If $k_R \neq k_S$, then the game returns 1, indicating that correctness does not hold. Otherwise, we increase the counter r and the oracle returns. We say that an RKE scheme RKE is correct if $\Pr[\mathsf{CORREC}_{\mathsf{RKE}}(\mathcal{A}) \Rightarrow 1] = 0$.

Recover Security. In similar vein as [DV19], we define recover security in game (q, ε)-RECOV in Fig. 9. The game proceeds in the same way as CORREC, but we change the winning condition. The adversary will win if sender and receiver are out-of-sync, but the receiver still accepts an encapsulation that was produced by the (honest) sender.

Sender and receiver can get out-of-sync by an impersonations. In particular, we say that $\mathcal{A}$ trivially impersonated the sender if it queries the Expose oracle, produces an encapsulation using the state stS and then receives this encapsulation. $\mathcal{A}$ may also non-trivially impersonate the sender without querying Expose. In either case, is will be set to false whenever Rcv successfully receives an encapsulation which was not delivered in order. For recover security, we now further check if the ciphertext input to `Rcv` appears in $\boldsymbol{cadk}$. From now on, all subsequent calls to `Rcv` should not accept an encapsulation produced by the game's sender state and we say that a scheme RKE is (q, ε)-$\mathsf{RECOV}_{\mathsf{RKE}}(\mathcal{A})$ if for any limited to q queries, the advantage is at most ε.

Game $\mathsf{CORREC}_{\mathsf{RKE}}(\mathcal{A})$ (dashed box) $\mathsf{RECOV}_{\mathsf{RKE}}(\mathcal{A})$ (solid box)

$\boldsymbol{cadk} \leftarrow [\cdot]$
$(s, r) \leftarrow (0, 0)$
$\text{is} \leftarrow \texttt{tru}$
$(\text{stS}, \text{stR}) \overset{\$}{\leftarrow} \mathsf{RKE.init}$
Invoke $\mathcal{A}$
Stop with 0

Oracle `Snd`(ad)

$(\text{stS}, c, k_S) \overset{\$}{\leftarrow} \mathsf{RKE.snd}(\text{stS}, \text{ad})$
$\boldsymbol{cadk}[s] \leftarrow (c, \text{ad}, k_S)$
$s \leftarrow s + 1$
Return (c, k_S)

Oracle `Expose`

Return (stS, stR)

Oracle `Rcv`(c, ad)

$(\text{stR}, k_R) \leftarrow \mathsf{RKE.rcv}(\text{stR}, c, \text{ad})$
If $k_R = \bot$:
  Return
$(c', \text{ad}', k_S) \leftarrow \boldsymbol{cadk}[r]$
If $(c, \text{ad}) \neq (c', \text{ad}')$:
  $\text{is} \leftarrow \texttt{fal}$
[solid box] If $\text{is} = \texttt{fal} \land \exists s^* : (c, \_, \_) = \boldsymbol{cadk}[s^*]$:
  Stop with 1
If $(c, \text{ad}) = (c', \text{ad}')$:
  [dashed box] If $\text{is} \land k_R \neq k_S$: Stop with 1
  $r \leftarrow r + 1$
Return

Oracle `RR`

$\text{stS} \overset{\$}{\leftarrow} \mathsf{RKE.rr}(\text{stS})$

Fig. 9. Games CORREC and RECOV for RKE scheme RKE defining correctness and recover security. The winning condition in the dashed box is only present in CORREC and the winning condition in the solid box is only present in RECOV.

B.1 Key Indistinguishability

The security game for key indistinguishability (KIND) of an RKE scheme is given in Fig. 10, where we assume the scheme is correct and robust. Intuitively, KIND security guarantees that the adversary cannot distinguish a key that is output by $\mathsf{RKE.snd}$ from a uniformly random key.

As in the previous games, we give the adversary access to oracles `Snd`, `Rcv`, `RR` in order to execute the algorithms of the RKE scheme, as well as oracles $\texttt{Expose}_S$ and $\texttt{Expose}_R$ which output the current sender and receiver state, respectively.

We additionally define a challenge oracle `ChallSnd` which takes an additional data string as input and returns an encapsulation c together with a key k. Depending on the bit b, k is either the real key (in game KIND^0) or a random key (in game KIND^1). Note that we do not define a `ChallRcv` oracle. Challenges may be received via the `Rcv` oracle.

In order to prevent trivial attacks, we introduce additional variables. In particular, we need to make sure that the adversary cannot decrypt a challenge encapsulation using an exposed receiver state. Therefore, the list $\boldsymbol{cad}$ stores the additional data provided by the adversary and encapsulations computed in `Snd` and `ChallSnd` queries. As in the correctness definition, we use counters s and r to keep track of all send queries and all in-order receive queries. We additionally store challenge encapsulations in a set $\boldsymbol{cc}$ and all successfully and in-order received encapsulations in a set $\boldsymbol{rcvd}$. We also track the impersonation status by the in-sync flag is, which is initially set to true, and exposures using flag xR, which is initially set to false. As long as sender and receiver are in-sync, $\texttt{Expose}_R$ can only be queried when all challenges are also received, i.e., $\boldsymbol{cc} \subseteq \boldsymbol{rcvd}$. In the same way, `ChallSnd` may not be queried if the sender and receiver are in-sync, but the receiver was exposed. Note that we cannot ensure key indistinguishability for this and future challenges. Once the sender is impersonated (i.e., sender and receiver are out-of-sync), we allow arbitrary queries to $\texttt{Expose}_R$ and `ChallSnd`.

Definition 11. *Consider the games* KIND^b *for* $b \in \{0, 1\}$ *in Fig. 10. We define the advantage of an adversary* $\mathcal{A}$ *against key indistiguishability of a ratcheted key exchange scheme* RKE *as*

$$\mathsf{Adv}^{\mathsf{KIND}}_{\mathcal{A},\mathsf{RKE}} := \left|\Pr[\mathsf{KIND}^0_{\mathsf{RKE}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathsf{KIND}^1_{\mathsf{RKE}}(\mathcal{A}) \Rightarrow 1]\right| \ .$$

```
Game KIND^b_RKE(A)
cad ← [·]
(cc, rcvd) ← (∅, ∅)
is ← tru
xR ← fal
(s, r) ← (0, 0)
(stS, stR) ←$ RKE.init
b' ←$ A
Stop with b'

Oracle Snd(ad)
(stS, c, k) ←$ RKE.snd(stS, ad)
cad[s] ← (c, ad)
s ← s + 1
Return (c, k)

Oracle RR
stS ←$ RKE.rr(stS)
Return

Oracle Expose_S
Return stS

Oracle ChallSnd(ad)
Require xR ≠ tru
(stS, c, k) ←$ RKE.snd(stS, ad)
cad[s] ← (c, ad)
cc ←∪ {s}
s ← s + 1
If b = 1: k ←$ K
Return (c, k)

Oracle Rcv(c, ad)
(stR, k) ← RKE.rcv(stR, c, ad)
If (c, ad) ≠ cad[r] ∧ k ≠ ⊥:
   is ← fal
If (c, ad) = cad[r] ∧ k ≠ ⊥:
   rcvd ←∪ {r}
   r ← r + 1
Return

Oracle Expose_R
If is = tru:
   Require cc ⊆ rcvd
   xR ← tru
Return stR
```

Fig. 10. Games KIND^b for $b \in \{0, 1\}$ and RKE scheme RKE with a *robust* correctness notion.

B.2 Authenticity

We define the security game AUTH which models authenticity in Fig. 11. Intuitively, an RKE scheme satisfies authenticity if for all in-sync sender and receiver states, the adversary cannot break the ordering of sent and received encapsulations, except for forgeries resulting from a trivial impersonation. We will analyze authenticity only for robust and correct schemes and we will also enforce recover security.

```
Game AUTH_RKE(A)
cad ← [·]
xS ← ∅
is ← tru
(s, r) ← (0, 0)
(stS, stR) ←$ RKE.init
Invoke A
Stop with 0

Oracle Snd(ad)
(stS, c, k) ←$ RKE.snd(stS, ad)
cad[s] ← (c, ad)
s ← s + 1
Return (c, k)

Oracle RR
stS ←$ RKE.rr(stS)
Return

Oracle Rcv(c, ad)
(stR, k) ← RKE.rcv(stR, c, ad)
If k ≠ ⊥ ∧ is = tru ∧ (c, ad) ≠ cad[r]:
   is ← fal
   If r ∉ xS: Stop with 1
If k ≠ ⊥:
   r ← r + 1
Return

Oracle Expose_S
xS ←∪ {s}
Return stS

Oracle Expose_R
Return stR
```

Fig. 11. Game AUTH for RKE scheme RKE with a *robust* correctness notion.

As in KIND, we provide the adversary with oracles `Snd`, `Rcv`, `RR`, $\texttt{Expose}_S$ and $\texttt{Expose}_R$. We also use the array $\boldsymbol{cad}$, counters s and r as well as the in-sync flag is. Recall that a trivial impersonation results from a call to $\texttt{Expose}_S$. In order to detect such an impersonation, we additionally introduce the set $\boldsymbol{xS}$ and we add the value of counter s at the time when $\texttt{Expose}_S$ is queried.

When `Rcv` is queried on (c, ad), we then check whether this is an impersonation attempt. For a successful impersonation, the input must be accepted by $\mathsf{RKE.rcv}$, i.e., $k \neq \bot$, and sender and receiver must have been in-sync before that query. Also the query (c, ad) must not be the next in-order tuple stored in $\boldsymbol{cad}[r]$. If this is the case, then this is indeed an impersonation and we set is $=$ `fal`. Now we check whether this is a trivial or non-trivial impersonation. If the value of counter r is not in the set $\boldsymbol{xS}$, then the adversary has come up with a non-trivial forgery and wins the authenticity game. Otherwise, if the value of counter r is in the set $\boldsymbol{xS}$, then this is a trivial impersonation and the game continues. However note that the adversary cannot win by providing a non-trivial forgery anymore, since sender and receiver will remain out-of-sync.

Definition 12. *Consider the game* AUTH *in Fig. 11. We define the advantage of an adversary $\mathcal{A}$ against authenticity of a ratcheted key exchange scheme* RKE *as*

$$\mathsf{Adv}^{\mathsf{AUTH}}_{\mathcal{A},\mathsf{RKE}} := \Pr[\mathsf{AUTH}_{\mathsf{RKE}}(\mathcal{A}) \Rightarrow 1] \ .$$

C Further Discussion on our Anonymity Defintion

In this section we provide some additional explanation on our anonymity definition. For each requirement we provide an example on how to break anonymity if we would allow such a query. As elaborated in Section 4.2, the requirements can explained by different relations: correctness ($\oplus$), sender state equality ($\triangleright$), receiver state equality ($\triangleleft$), matching states ($\diamond$) and impersonations (i). At the end of this section, we also argue why we require authenticity in the first place.

Queries to `Snd`*.* For oracle `Snd`, we have the following requirement.

- Line 42 ($\oplus$): If there has not been an impersonation yet, but a query to $\texttt{ChallExpose}_R$ has been issued, then we cannot allow queries to `Snd` since the challenge receiver state can be used to decrypt the ciphertext which will be successful only if $b = 0$.

Note that we allow queries to `Snd` after $\texttt{ChallExpose}_R$ in case there has been an impersonation. In this case, the exposed challenge receiver state does not leak b.

Queries to `ChallSnd`*.* The restrictions are similar to those of `Snd`.

- Line 49 ($\oplus$): If there has not been an impersonation in $\mathsf{U\text{-}ANON}_0$, we require that $\texttt{Expose}_R$ or $\texttt{ChallExpose}_R$ have not been queried as well. Otherwise, the adversary can try to decrypt the challenge ciphertext with the exposed receiver state. If decryption is successful, it knows that $b = 0$.

Note that we only have this requirement if $\mathrm{imp}_0 =$ `fal`. As soon as $\mathsf{U\text{-}ANON}_0$ is impersonated, the challenge ciphertext cannot be decrypted by an exposed (challenge) receiver state, independent of b.

Queries to $\texttt{Expose}_S$*.* We have to prevent several trivial attacks for sender exposures.

- Line 14 ($\triangleright$): In most cases, we need to disallow sender exposures whenever there was a query to $\texttt{ChallExpose}_S$. However, note that we can allow sender exposures after $\texttt{ChallExpose}_S$ if there has been an update in between or a query to `ChallSnd`. Due to the progression of the sender state, the output in ANON^0 should be indistinguishable from the output in ANON^1.
- Lines 15-16 ($\triangleright$): We cannot allow two subsequent queries to $\texttt{Expose}_S$ if there has been a query to `ChallSnd` in between, without any further update, since the sender states would be the same in $b = 0$, but not in $b = 1$.
- Lines 17-18 ($\diamond$): We do not allow sender exposures if $\texttt{ChallExpose}_R$ has been queried and there has not been an impersonation yet, since the adversary could check whether the exposed states belong together.

One could assume that we also have to prevent queries to $\texttt{Expose}_S$ after a query to $\texttt{ChallExpose}_R$ if only $\mathsf{U\text{-}ANON}_1$ has been impersonated. However, note that an impersonation exclusively in $\mathsf{U\text{-}ANON}_1$ can only happen after a `ChallSnd` query which in turn disallows a query to $\texttt{ChallExpose}_R$.

Queries to $\texttt{ChallExpose}_S$. We need the following restrictions for oracle $\texttt{ChallExpose}_S$.

- Lines 32-33 (▷): We cannot allow a query to $\texttt{ChallExpose}_S$ if there has been a query to $\texttt{Expose}_S$ or $\texttt{ChallExpose}_S$ before (without an update in between). Otherwise, the adversary can simply compare the outputs.
- Lines 34-35 (⋄): As long as there has not been an impersonation yet, we do not allow queries to $\texttt{ChallExpose}_S$ after a receiver exposure via $\texttt{Expose}_R$ or $\texttt{ChallExpose}_R$ since in ANON^0, sender and receiver states belong together, which they do not in ANON^1.

Queries to $\texttt{Expose}_R$. For oracle $\texttt{Expose}_R$, we need the following requirements.

- Line 23 (i): Recall that variable unique is set to $\texttt{fal}$ if the impersonation state in $\mathsf{U\text{-}ANON}_0$ cannot be uniquely determined. An example for this case is the following sequence of queries: $\texttt{Expose}_S$, $\texttt{ChallSnd}$, $\texttt{RR}$, $\texttt{Expose}_S$ and an impersonation attempt with one of the exposed sender states. Assume the adversary used the output of the second sender exposure query to impersonate. In ANON^1, the impersonation is successful, but not in ANON^0, which means we have to disallow a receiver exposure. On the other hand, assume the adversary used the output of the first sender exposure query to impersonate. Then in both ANON^0 and ANON^1, the impersonation is successful and a receiver exposure should be allowed. However, in ANON^1 we cannot know whether impersonation was attempted with the sender state of the first or the second exposure query, thus we have disallow receiver exposures in the first place.
- Line 24 (◁): After a query to $\texttt{ChallExpose}_R$, we need to disallow any subsequent queries to $\texttt{Expose}_R$ since otherwise the adversary can determine b by simply comparing the exposed states.
- Line 25 (⋄): For receiver exposures, we require that the sequence of queries so far either results in an impersonation in both utopian games or in none of the utopian games, i.e., $\mathrm{imp}_0 = \mathrm{imp}_1$. If an impersonation happens only in one of the games, then the adversary knows the corresponding sender state that it used for an impersonation attempt and thus it can compare whether the two states belong together. Examples for unallowed sequences are ($\texttt{Expose}_S$, $\texttt{ChallSnd}$, $\texttt{Rcv}$ to impersonate, $\texttt{Expose}_R$) and ($\texttt{ChallExpose}_S$, $\texttt{Rcv}$ to impersonate, $\texttt{Expose}_R$).
- Line 27 (⊕): If there has not been an impersonation yet, we require that all challenge ciphertexts have been received before querying $\texttt{Expose}_R$. Otherwise, the exposed receiver state will allow to decrypt a challenge ciphertext in ANON^0, but not in ANON^1.
- Line 28 (⋄): After a query to $\texttt{ChallExpose}_S$, we cannot allow receiver exposures that would allow to compare whether the two states belong together.

Queries to $\texttt{ChallExpose}_R$. Receiver exposures via $\texttt{ChallExpose}_R$ need to enforce the following requirements.

- Line 79 (◁): We do not allow queries to $\texttt{ChallExpose}_R$ when the receiver state has already been exposed either via $\texttt{Expose}_R$ or $\texttt{ChallExpose}_R$, since this would allow to compare the outputs.
- Line 80 (⋄): After a sender exposure via $\texttt{Expose}_S$ or $\texttt{ChallExpose}_S$, we cannot allow a challenge exposure of the receiver at the same point in time.
- Line 81 (⋄): We cannot allow queries to $\texttt{ChallExpose}_R$ if there has been an impersonation in $\mathsf{U\text{-}ANON}_0$. Assume there has been an impersonation in $\mathsf{U\text{-}ANON}_0$, then the adversary knows the corresponding sender state and can check if the exposed receiver state actually belongs to that sender state.
- Line 82 (⊕): If there has not been any impersonation yet, we require that all ciphertext (challenges and non-challenges) have been received before calling $\texttt{ChallExpose}_R$, otherwise the adversary could simply check if decryption is successful.

Finally, we justify that we require authenticity in our anonymity definition.

Claim. For any adversary that breaks authenticity (cf. Definition 12) of a ratcheted key exchange scheme RKE, there exists an adversary that breaks anonymity (cf. Definition 1) of that scheme.

Proof. Recall that authenticity means that an adversary cannot perform a non-trivial impersonation, i.e., it cannot compute a ciphertext that will be received successfully without the knowledge of the sender state. Then an adversary against anonymity can query the challenge oracle for receiver exposures $\texttt{ChallExpose}_R$ and check if the given receiver state successfully decrypts the ciphertext of that non-trivial impersonation.

D Omitted Definitions and Proofs from Section 5

In Appendix D.1 we give the proofs for the properties of rPKE we use to prove ANON of RKE. In Appendix D.2 we give the security properties of urPKE we use to prove KIND and ANON of RKE. In Appendix D.3 we finally give the construction of $\mathsf{urPKE}_{\mathsf{EG}}$ and all related proofs.

D.1 The Homomorphic property and Indistinguishability of Encryptions of $\mathsf{rPKE}_{\mathsf{EG}}$

Proof (of Lemma 1). We first show the homomorphic property of $\mathsf{rPKE}_{\mathsf{EG}}$. To this end, observe that for an encryption key $\mathrm{ek} = (c_0, k_0)$, we have that

$$\begin{aligned}\mathsf{rPKE.enc}(\mathrm{ek}, m_1; r_1) \otimes \mathsf{rPKE.enc}(\mathrm{ek}, m_2; r_2) &= (\mathrm{ek}_0^{r_1}, \mathrm{ek}_1^{r_1} \cdot m_1) \otimes (\mathrm{ek}_0^{r_2}, \mathrm{ek}_1^{r_2} \cdot m_2) \\ &= (\mathrm{ek}_0^{r_1+r_2}, \mathrm{ek}_1^{r_1+r_2} \cdot m_1 \cdot m_2) \\ &= \mathsf{rPKE.enc}(\mathrm{ek}, m_1 \otimes m_2; r_1 \oplus r_2) \ .\end{aligned}$$

To show indistinguishability of randomized encryptions we first give a reduction of a multi challenge version of DDH to standard DDH. Then we reduce the advantage of distinguishing D_1 and D_2 to the multi challenge version of DDH. We define the multi challenge version of DDH such that it should be hard to distinguish between distributions

$$\begin{aligned}D' &= (g^{x_0}, \dots, g^{x_\ell}, Y := g^y, Y^{x_0}, \dots, Y^{x_\ell}) \\ D^* &= (g^{x_0}, \dots, g^{x_\ell}, Y, g^{z_0}, \dots, g^{z_\ell})\end{aligned}$$

where $x_0, \dots, x_\ell, y, z_0, \dots, z_\ell \xleftarrow{\$} \mathbb{Z}_p$. We have

$$|\Pr[\mathcal{D}(D')] - \Pr[\mathcal{D}(D^*)]| \le \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}_1,\mathbb{G}}$$

where we construct $\mathcal{B}_1$ as follows: $\mathcal{B}_1$ inputs challenge (X, Y, Z). It chooses $r_i \xleftarrow{\$} \mathbb{Z}_p^\ell$ and forwards $(X^{r_0}, \dots, X^{r_0\ell},\ Y, Z^{r_0}, \dots, Z^{r_0\ell})$ to the distinguisher $\mathcal{D}$. It returns whatever $\mathcal{D}$ returns. Note that if (X, Y, Z) is a DDH tuple, then $\mathcal{B}_1$ outputs distribution D'. If (X, Y, Z) is not a DDH tuple, then it outputs distribution D^*.

Next, we prove indistinguishability of randomized encryptions. We need to show that for all $m_0, \dots, m_{2\ell}$ the following two distributions are indistinguishable:

$$\begin{aligned}D_1 &= (\mathrm{ek}, c_0, \dots, c_\ell, \mathrm{ek}', d_0, \dots, d_\ell) \\ &= (\mathrm{ek}_0, \mathrm{ek}_1, c_0^0, c_0^1, \dots, c_\ell^0, c_\ell^1, \mathrm{ek}_0', \mathrm{ek}_1', d_0^0, d_0^1, \dots, d_\ell^0, d_\ell^1) \\ &= (g^r, g^{rx}, (g^{rs_0}, g^{rs_0x} \cdot m_0), \dots, (g^{rs_\ell}, g^{rs_\ell x} \cdot m_\ell), g^{rr'}, g^{rr'x}, \\ &\qquad (g^{rs_0+rr's_0'}, g^{(rs_0+rr's_0')x} \cdot m_0), \dots, (g^{rs_\ell+rr's_\ell'}, g^{(rs_\ell+rr's_\ell')x} \cdot m_\ell)) \\ D_2 &= (\mathrm{ek}, c_0, \dots, c_\ell, \hat{\mathrm{ek}}, e_0, \dots, e_\ell) \\ &= (\mathrm{ek}_0, \mathrm{ek}_1, c_0^0, c_0^1, \dots, c_\ell^0, c_\ell^1, \hat{\mathrm{ek}}_0, \hat{\mathrm{ek}}_1, e_0^0, e_0^1, \dots, e_\ell^0, e_\ell^1) \\ &= (g^r, g^{rx}, (g^{rs_0}, g^{rs_0x} \cdot m_0), \dots, (g^{rs_\ell}, g^{rs_\ell x} \cdot m_\ell), g^u, g^{ux'}, \\ &\qquad (g^{uv_0}, g^{uv_0x'} \cdot m_{\ell+1}), \dots, (g^{uv_\ell}, g^{uv_\ell x'} \cdot m_{2\ell}))),\end{aligned}$$

where $r, r', s_0, \dots, s_\ell, s_0', \dots, s_\ell', u, v_0, \dots, v_\ell, x, x' \xleftarrow{\$} \mathbb{Z}_p$.

We replace rr' in D_1 with a new variable u and $r's_i$ with a new variable z_i. We get

$$\begin{aligned}D_1 = (&g^r, g^{rx}, (g^{z_0}, g^{z_0x} \cdot m_0), \dots, (g^{z_\ell}, g^{z_\ell x} \cdot m_\ell), g^u, g^{ux}, \\ &(g^{z_0+us_0'}, g^{(z_0+us_0')x} \cdot m_0), \dots, (g^{z_\ell+us_\ell'}, g^{(z_\ell+us_\ell')x} \cdot m_\ell))\end{aligned}$$

Note that s_i' only appears in the last 2ℓ terms and in all cases it appears as us'. As s_i' is uniformly chosen at random, the term $z_i + us_i'$ is distributed as a uniformly random element w_i. We get

$$\begin{aligned}D_1 = (&g^r, g^{rx}, (g^{z_0}, g^{z_0x} \cdot m_0), \dots, (g^{z_\ell}, g^{z_\ell x} \cdot m_\ell), g^u, g^{ux}, \\ &(g^{w_0}, g^{w_0x} \cdot m_0), \dots, (g^{w_\ell}, g^{w_\ell x} \cdot m_\ell))\end{aligned}$$

Now we use the multi challenge DDH assumption to replace all group elements which are g^x raised to some other element from $\mathbb{Z}_p$.

$$D' = (g^r, g^{a_0}, (g^{z_0}, g^{a_1} \cdot m_0), \ldots, (g^{z_\ell}, g^{z_\ell a_{\ell+1}} \cdot m_\ell), g^u, g^{u a_{\ell+2}},$$
$$(g^{w_0}, g^{a_{\ell+3}} \cdot m_0), \ldots, (g^{w_\ell}, g^{w_\ell a_{2\ell+3}} \cdot m_\ell))$$

We have

$$|\Pr[\mathcal{D}(D_1)] - \Pr[\mathcal{D}(D')]| \leq \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B}_2,\mathbb{G}}$$

where we construct $\mathcal{B}_2$ as follows: $\mathcal{B}_2$ inputs challenge $(X_0, \ldots, X_{2\ell+3}, Y, Z_0, \ldots, Z_{2\ell+3})$. It chooses $(r_0, \ldots, r_{2\ell+3}) \xleftarrow{\$} \mathbb{Z}_p^{2\ell+3}, (m_0, \ldots, m_\ell) \xleftarrow{\$} \mathcal{M}^\ell$ and forwards $(g^{r_0}, g^{a_0}, (g^{r_1}, X_0^{r_1} \cdot m_0), \ldots, (g^{r_{\ell+1}}, X_{\ell+1}^{r_{\ell+1}}), g^{r_{\ell+2}}, X_{\ell+2}^{r_{\ell+2}}, (g^{r_{\ell+3}}, X_{\ell+3}^{r_{\ell+3}}), \ldots, (g^{r_{2\ell+3}}, X_{2\ell+3}^{r_{2\ell+3}}))$ to the distinguisher $\mathcal{D}$. It returns whatever $\mathcal{D}$ returns. Note that if $(X_0, \ldots, X_{2\ell+3}, Y, Z_0, \ldots, Z_{2\ell+3})$ is a multi challenge DDH tuple, then $\mathcal{B}_2$ outputs distribution D_1. Else adversary $\mathcal{B}_2$ outputs distribution D'.

D.2 Omitted Security Definitions for urPKE

Below we give the formal security definitions for indistinguishability and anonymity of an updatable public key encryption scheme.

Definition 13 (Indistinguishability of Ciphertexts). *Consider the games in Fig. 12. We define the advantage of an adversary $\mathcal{A}$ against key indistinguishability of a public key encryption scheme* urPKE *as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{A},\mathsf{urPKE}} \coloneqq \left|\Pr[\mathsf{IND\text{-}C}^0_{\mathsf{urPKE}}(\mathcal{A}) \to 1] - \Pr[\mathsf{IND\text{-}C}^1_{\mathsf{urPKE}}(\mathcal{A}) \to 1]\right| .$$

Game $\mathsf{IND\text{-}C}^b_{\mathsf{urPKE}}(\mathcal{A})$	**Oracle** $\mathtt{ChallSnd}(m_0)$ //only once
$(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{urPKE.gen}$	$m_1 \xleftarrow{\$} \mathcal{M}$
$\boldsymbol{mc} \leftarrow [\cdot]$	$c \xleftarrow{\$} \mathsf{urPKE.enc}(\mathsf{ek}, m_b)$
$b' \xleftarrow{\$} \mathcal{A}^{\mathtt{ChallSnd},\mathtt{RR}}(\mathsf{ek})$	$\boldsymbol{mc}.\mathsf{append}(m_0, c)$
Stop with b'	Return c
Oracle $\mathtt{RR}$	**Oracle** $\mathtt{Dec}(c)$
$\mathsf{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathsf{ek})$	If $(m^*, c) \in \boldsymbol{mc}$:
Return ek	Return m^*
	Return $\mathsf{urPKE.dec}(\mathsf{dk}, c)$

Fig. 12. Security games $\mathsf{IND\text{-}C}^b$ for an updatable and randomizable PKE scheme urPKE.

Definition 14 (Anonymity). *Consider games* $\mathsf{ANON}^b_{\mathsf{urPKE}}$ *for* $b \in \{0, 1\}$ *in Fig. 13. We define the advantage of an adversary $\mathcal{A}$ against anonymity of a public key encryption scheme* urPKE *as*

$$\mathsf{Adv}^{\mathsf{ANON}}_{\mathcal{A},\mathsf{urPKE}} \coloneqq \left|\Pr[\mathsf{ANON}^0_{\mathsf{urPKE}}(\mathcal{A}) \to 1] - \Pr[\mathsf{ANON}^1_{\mathsf{urPKE}}(\mathcal{A}) \to 1]\right| .$$

D.3 Our Construction of urPKE

Our ElGamal-based construction of an updatable and randomizable PKE scheme $\mathsf{urPKE}_{\mathsf{EG}}$ is given in Fig. 14. In Appendix D.3 we prove instance independence and indistinguishability of randomizations. Then, in Theorems 2 and 3 we prove indistinguishability and anonymity of the scheme.

Theorem 2 (**IND-C security of** $\mathsf{urPKE}_{\mathsf{EG}}$). *Let* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *be a random oracle. For any adversary $\mathcal{A}$ against* IND-C *security of* $\mathsf{urPKE}_{\mathsf{EG}}$*, there exists an adversary $\mathcal{B}$ against* GDH *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{A},\mathsf{urPKE}} \leq \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathbb{G}} + \frac{1}{2^\lambda} .$$

Proof. We start by a sequence of games from G_0 to G_3 given in Fig. 15.

```
Game ANON^b_urPKE(A)
  mc ← [·]
  (ek, dk) ← urPKE.gen
  b' ←$ A(ek)
  Stop with b'

Oracle Expose_S
  Return ek

Oracle RR
  ek ←$ urPKE.rr(ek)
  Return

Oracle Dec(c)
  If (m*, c) ∈ mc:
    Return m*
  Return urPKE.dec(dk, c)

Oracle ChallSnd(m)        //only once
  c_0 ←$ urPKE.enc(ek, m)
  (ek', _) ←$ urPKE.gen
  c_1 ←$ urPKE.enc(ek', m)
  mc.append(m_0, c)
  Return c_b
```

Fig. 13. Security games ANON^b for an updatable and randomizable PKE urPKE.

```
Proc urPKE.gen
  x, r ←$ Z_p
  ek ← (g^r, g^{xr}); dk ← x
  Return (ek, dk)

Proc urPKE.enc(ek, m)
  Parse ek as (ek_0, ek_1)
  s ←$ Z_p
  (c_0, c_1) ← (ek_0^s, H(ek_0^s, ek_1^s) ⊕ m)
  Return (c_0, c_1)

Proc urPKE.dec(dk, c)
  Parse c as (c_0, c_1)
  m ← H(c_0, c_0^dk) ⊕ c_1
  Return m

Proc urPKE.rr(ek)
  Parse ek as (ek_0, ek_1)
  r' ←$ Z_p
  Return (ek_0^{r'}, ek_1^{r'})

Proc urPKE.nextDk(dk, r)
  dk' ← dk · r
  Return dk'

Proc urPKE.nextEk(ek, r)
  Parse ek as (ek_0, ek_1)
  s ←$ Z_p
  ek' ← (ek_0^s, ek_1^{rs})
  Return ek'
```

Fig. 14. Updatable and randomizable PKE scheme $\mathsf{urPKE_{EG}}$.

Experiment Exp_0. This game is equivalent to $\mathsf{IND\text{-}C}^0$.

Experiment Exp_1. In this game we exclude the case that the adversary guesses the output to any input of the random oracle without querying that input. We have $\left|\Pr\left[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \frac{1}{2^\lambda}$.

Experiment Exp_2. In a call to oracle `ChallSnd`, we replace the input to the random oracle with uniformly random values from the input space. To show that no adversary can distinguish games G_1 and G_2, we now show that any such adversary $\mathcal{A}$ can be turned into an adversary $\mathcal{B}$ against GDH in $\mathbb{G}$.

Adversary $\mathcal{B}$ is called on distribution $D := (X, Y) \in \mathbb{G}^2$.

To simulate the game for $\mathcal{A}$, adversary $\mathcal{B}$ embeds distribution D as follows. At the beginning of the experiment, $\mathcal{B}$ samples random $r \xleftarrow{\$} \mathbb{Z}_p$ and sets $\mathsf{ek} \leftarrow (g^r, X^r)$. On a call to oracle `ChallSnd`, adversary $\mathcal{B}$ sets $(c_1, k) \leftarrow (Y^r, Z)$, where $Z \xleftarrow{\$} \mathbb{G}$. Since the output of the random oracle is unpredictable, $\mathcal{A}$ must call the random oracle on the correct input to the random oracle in order to distinguish the two distributions. From that call to the random oracle, adversary $\mathcal{B}$ can extract a solution to the GDH problem.

To simulate the random oracle, adversary $\mathcal{B}$ checks for every input (Y, Z) to the random oracle, whether $\mathtt{DDH}(X, Y, Z) = 1$. If so, adversary $\mathcal{B}$ returns Z to the GDH experiment.

To simulate queries to oracle $\mathtt{Dec}(c)$, adversary $\mathcal{B}$ does the following. It parses $(c_0, c_1) \leftarrow c$. Then it searches the list of random oracle queries until it finds an entry (Y, Z) s.t. $\mathtt{DDH}(c_1, X, Z) = 1$. If the adversary finds such an entry, it outputs $m \leftarrow c_2 \oplus \mathsf{H}(Y, Z)$. Otherwise, it returns a random bitstring. If a ciphertext is well-distributed, then it must hold that $\mathtt{DDH}(c_1, g^x, k) = 1$. Thus, $\left|\Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \mathsf{Adv}_{\mathbb{G}}^{\mathsf{GDH}}$.

Experiment Exp_3. In this game we replace c_2 with uniform randomness. Since c_2 is now independent of the underlying message, we have $\Pr[{\mathsf{G}_3}^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{IND\text{-}C}^1_{\mathcal{A}} \Rightarrow 1]$.

So in total, we have

$$\left|\Pr\left[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \mathsf{Adv}_{\mathcal{B},\mathbb{G}}^{\mathsf{GDH}} + \frac{1}{2^\lambda} \ ,$$

which concludes the proof.

Games $\mathsf{G}_0 = \mathsf{IND\text{-}C}^0, \mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3 = \mathsf{IND\text{-}C}^1$

$x, r \xleftarrow{\$} \mathbb{Z}_p$
$\boldsymbol{mc} \leftarrow [\cdot]$
$\mathrm{ek} \leftarrow (g^r, g^{xr})$; $\mathrm{dk} \leftarrow x$
$b' \xleftarrow{\$} \mathcal{A}^{\mathtt{ChallSnd},\mathtt{RR},\mathtt{Dec}}(\mathrm{ek})$
Stop with b'

Random Oracle $\mathsf{H}(Y, Z)$

if $h[(Y, Z)] \neq \bot$
  ABORT //G_1-G_3
  Return $h[(Y, Z)]$
$c \xleftarrow{\$} \{0,1\}^\lambda$
$h[(Y, Z)] \leftarrow c$
Return c

Oracle $\mathtt{RR}$

$\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$
Return ek

Oracle $\mathtt{ChallSnd}(m_0)$ //only once

$m_1 \xleftarrow{\$} \mathcal{M}$; $s \xleftarrow{\$} \mathbb{Z}_p$
$(\mathrm{ek}_0, \mathrm{ek}_1) \leftarrow \mathrm{ek}$
$c_2 \leftarrow \mathsf{H}(\mathrm{ek}_0^s, \mathrm{ek}_1^s) \oplus m_b$ //G_0-G_2
$c_2 \leftarrow \mathsf{H}(\xleftarrow{\$} \mathbb{G} \times \mathbb{G}) \oplus m_b$ //G_2-G_3
$c_2 \xleftarrow{\$} \{0,1\}^\lambda$ //G_3
$\boldsymbol{mc}.\mathsf{append}(m_0, (c_1, c_2))$
Return (c_1, c_2)

Oracle $\mathtt{Dec}(c)$

If $(m^*, c) \in \boldsymbol{mc}$:
  Return m^*
Parse c as (c_0, c_1)
$m \leftarrow \mathsf{H}(c_0, c_0^{\mathrm{dk}}) \oplus c_1$
$m \leftarrow h[c_0, c_0^{\mathrm{dk}}] \oplus c_1$ //G_1-G_3
$m \leftarrow h[c_0, c_0^{\mathrm{dk}}] \oplus c_1$ //G_2-G_3
Return m

Fig. 15. Games G_0-G_3 for the proof of Theorem 2.

Theorem 3 (ANON security of $\mathsf{urPKE_{EG}}$). *Let* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *be a random oracle. For any adversary* $\mathcal{A}$ *against* ANON *security of* $\mathsf{urPKE_{EG}}$*, there exist an adversary* $\mathcal{B}$ *against* GDH.

$$\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{A},\mathsf{urPKE}} \leq \mathsf{Adv}^{\mathsf{GDH}}_{\mathcal{B},\mathbb{G}} + \frac{1}{2^\lambda} .$$

The proof for this Theorem follows the same steps as the proof of Theorem 2.

Lemma 3. *Scheme* $\mathsf{urPKE_{EG}}$ *is instance-independent.*

Proof. Let $(\mathrm{ek}_1, \mathrm{dk}_1) \xleftarrow{\$} \mathsf{urPKE.gen}, r, s \xleftarrow{\$} \mathcal{R}, \mathrm{ek}_2 \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}_1, r), \mathrm{dk}_2 \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}_1, r)$ be random, then by definition of $\mathsf{urPKE.gen}$,

$$\mathcal{D}(\mathrm{ek}_1, \mathrm{dk}_1, \mathrm{ek}_2, \mathrm{dk}_2) = \mathcal{D}((c_1, k_1), \mathrm{dk}_1, \mathrm{ek}_2, \mathrm{dk}_2) = \mathcal{D}((g^t, g^{tx}), x, \mathrm{ek}_2, \mathrm{dk}_2) .$$

By definition of $\mathsf{urPKE.nextEk}$ and $\mathsf{urPKE.nextDk}$,

$$\mathcal{D}((g^t, g^{tx}), x, \mathrm{ek}_2, \mathrm{dk}_2) = \mathcal{D}((g^t, g^{tx}), x, (g^{ts}, g^{txrs}), x \cdot r) .$$

Let $x' \leftarrow x \cdot r, t' \leftarrow t \cdot s$ then x' and t' are random values independent of x and t, thus

$$\mathcal{D}((g^t, g^{tx}), x, (g^{ts}, g^{txrs}), x \cdot r) = \mathcal{D}((g^t, g^{tx}), x, (g^{t'}, g^{t'x'}), x') = \mathcal{D}((g^t, g^{tx}), x, \mathrm{ek}_2, \mathrm{dk}_2) .$$

Lemma 4. *For any adversary* $\mathcal{A}$ *against indistinguishability of randomizations of* $\mathsf{urPKE_{EG}}$*, there exists an adversary* $\mathcal{B}$ *against* DDH *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{A},\mathsf{urPKE_{EG}}} \leq \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{B},\mathbb{G}} .$$

Proof. We need to show that the following two distributions are indistinguishable:

$$D_1 = (g^r, g^{xr}, g^{rr'}, g^{rr'x})$$
$$D_2 = (g^r, g^{xr}, g^s, g^t)$$

where $r, r', s, t, x \xleftarrow{\$} \mathbb{Z}_p$.

Let $h = g^r$, then

$$(g^r, g^{xr}, g^{rr'}, g^{rr'x}) = (h, h^x, h^{r'}, h^{r'x})$$

By DDH

$$(h, h^x, h^{r'}, h^{r'x}) \approx (h, h^x, h^{r'}, h^{r''}),$$

where $r'' \xleftarrow{\$} \mathbb{Z}_p$. So

$$(h, h^x, h^{r'}, h^{r''}) = (g^r, g^{xr}, g^{rr'}, g^{r''r}) = (g^r, g^{xr}, g^s, g^t) .$$

E Omitted Definitions and Proofs for Updatable and Randomizable Signatures

In Appendix E.1 we formally define one-time unforgeability and other properties tailored to our proofs of authenticity and recover security of RKE. The proof for strong unforgeability of urSIG comes in multiple steps, which we depict in Fig. 16.

$$\begin{array}{c}\mathsf{OT\text{-}EUF\text{-}CMA}_{\text{Lamport}}\\ +\\ \text{Pairing}\end{array} \xrightarrow{\text{Lemma 6}} \begin{array}{c}\mathsf{OT\text{-}EUF\text{-}CMA}_{\mathsf{uSIG}}\\ +\\ \mathsf{rPKE}\end{array} \xrightarrow{\text{Lemma 9}} \begin{array}{c}\mathsf{OT\text{-}EUF\text{-}CMA}_{\mathsf{urSIG}}\\ +\\ \mathsf{SUF\text{-}CMA}_{\mathsf{SIG}}\end{array} \xrightarrow{\text{Lemma 11}} \mathsf{OT\text{-}SUF\text{-}CMA}_{\mathsf{urSIG}}$$

Fig. 16. Overview of the proofs related to unforgeability of urSIG.

E.1 Omitted Definitions

In Definition 15 we give two standard unforgeability definitions for signature schemes, in Definition 17 and Definition 18 we respectively give definitions of properties tailored to our proofs for authenticity and recover security of RKE.

Definition 15 (One-Time Unforgeability). *Consider the* OT-EUF-CMA *and* OT-SUF-CMA *security games for signature schemes in Fig. 17. We define the advantage of an adversary $\mathcal{A}$ against* OT-EUF-CMA *and* OT-SUF-CMA *security of an updatable and randomizable one-time signature scheme as*

$$\mathsf{Adv}^{\mathsf{OT\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{urSIG}} := \Pr[\mathsf{OT\text{-}EUF\text{-}CMA}_{\mathsf{urSIG}}(\mathcal{A}) \Rightarrow 1]$$

and

$$\mathsf{Adv}^{\mathsf{OT\text{-}SUF\text{-}CMA}}_{\mathcal{A},\mathsf{urSIG}} := \Pr[\mathsf{OT\text{-}SUF\text{-}CMA}_{\mathsf{urSIG}}(\mathcal{A}) \Rightarrow 1] \ .$$

```
Game OT-EUF-CMA, [OT-SUF-CMA]
(vk, sk) ← urSIG.gen
q_S ← 0
S ← ∅
(m*, σ*) ← A^{Sign(·)}(vk)
If urSIG.vfy(vk, m*, σ*) = 1 ∧ q_S ≤ 1:
[   If (m*, σ*) ∉ S:
        Return 1
    Return 0 ]
    If (m*, _) ∉ S:
        Return 1
Return 0

Oracle Sign(m)
σ ← urSIG.sig(sk, m)
S ←∪ {(m, σ)}
q_S ← q_S + 1
Return σ
```

Fig. 17. Security notions OT-EUF-CMA and OT-SUF-CMA for signature schemes.

Definition 16 (Indistinguishability of Randomizations). *We require that a pair of encryption key and ciphertext that has been randomized via* urPKE.rr *is indistinguishable from a freshly generated encryption key and ciphertext. More formally, we define the advantage of a distinguisher $\mathcal{D}$ for arbitrary $m_0, m_1 \in \mathcal{M}$ as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{D},\mathsf{urPKE}} := |\Pr[\mathcal{D}(\mathrm{ek}, c, \mathrm{ek}_0, c_0) \Rightarrow 1] - \Pr[\mathcal{D}(\mathrm{ek}, c, \mathrm{ek}_1, c_1) \Rightarrow 1]| \ ,$$

where $(\mathrm{ek}, \_) \xleftarrow{\$} \mathsf{urPKE.gen}$, $c \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, m_0)$, $(\mathrm{ek}_0, c_0) \leftarrow \mathsf{urPKE.rr}(\mathrm{ek}, c)$, $(\mathrm{ek}_1, \_) \xleftarrow{\$} \mathsf{urPKE.gen}$, $c_1 \xleftarrow{\$} \mathsf{urPKE}(\mathrm{ek}_1, m_1)$.

Definition 17. *We say a signature scheme is verification key randomization smooth if there exists an algorithm* $\mathsf{findUpdate}(\mathrm{vk}_{\mathrm{next}}) \to (\mathrm{vk}, R)$ *which on input a target verification key* $\mathrm{vk}_{\mathrm{next}}$ *sampled uniformly at random from the support of* $\mathsf{urSIG.gen}$ *outputs another verification key* vk *from the support of* $\mathsf{urSIG.gen}$ *and a randomness* R *from the randomization space s.t.* $\mathrm{vk}_{\mathrm{next}} = \mathsf{urSIG.nextVk}(\mathrm{vk}, R)$ *and* R *is uniformly distributed in the randomization space and* vk *is uniformly distributed in the support of* $\mathsf{urSIG.gen}$.

Definition 18. *We say a updatable signature scheme is* ε*-unverifiable under random verification keys if for all* $(\mathrm{vk}, \mathrm{sk}) \in \mathsf{uSIG.gen}, m \in \{0,1\}^*$,

$$\Pr\left[\mathsf{uSIG.vfy}(\mathrm{vk}', m, \sigma) = 1 \,\middle|\, \begin{array}{l}\sigma \xleftarrow{\$} \mathsf{uSIG.sig}(\mathrm{sk}, m)\\ (\mathrm{vk}', \_) \xleftarrow{\$} \mathsf{uSIG.gen}\end{array}\right] \leq \varepsilon \ .$$

E.2 Omitted Theorems and Proofs for uSIG

On the way of proving the security properties defined in Appendix E.1 for urSIG, we give here the proofs for uSIG and then give in the next subsection generic reductions of the security of urSIG to the security of uSIG.

Lemma 5. *Signature scheme* uSIG *is verification key randomization smooth.*

Proof. We define algorithm $\mathsf{findUpdate}(\mathrm{vk}_{\mathrm{next}}) \to (\mathrm{vk}, R)$ as follows. Algorithm findUpdate samples R uniformly from the randomization space, it takes $\mathrm{vk}_{\mathrm{next}}$ and computes $\mathrm{vk}_{i,b} \leftarrow \mathrm{vk}_{\mathrm{next}\,i,b} \cdot e(R_{i,b}, g_2)^{-1}$ for all $b \in \{0,1\}, i \in \ell$. It returns (R, vk). Clearly, R is uniformly distributed in the randomization space. Therefore, if $\mathrm{vk}_{\mathrm{next}}$ is uniformly distributed in the support of uSIG.gen, then vk is also uniformly distributed in the support of uSIG.gen.

Lemma 6 (OT-EUF-CMA security of uSIG). *Let* H *be a random oracle. For any adversary* $\mathcal{A}$ *against* OT-EUF-CMA *of* uSIG*, there exists an adversary* $\mathcal{B}$ *against the fixed argument pairing inversion problem* FAPI-2 *such that*

$$\mathsf{Adv}^{\mathsf{OT\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{uSIG}} \leq 2\ell \cdot \mathsf{Adv}^{\mathsf{FAPI\text{-}2}}_{\mathcal{B},\mathbb{G}} + \frac{(q_H + q_S)}{2^\lambda} \ .$$

Proof. Let EUF-CMA′ be the same experiment as EUF-CMA with the only difference that EUF-CMA′ aborts if the hash function outputs a collision. Since H is modeled as a random oracle, this happens with probability at most $\frac{(q_H+q_S)}{2^\lambda}$.

Let $\mathcal{A}$ be an adversary against the one-time EUF-CMA′ security of uSIG. We show how to construct an adversary $\mathcal{B}$ which internally runs $\mathcal{A}$ and breaks the fixed argument pairing inversion problem FAPI-2.

The proof follows the same proof structure as a proof for Lamport signatures. In the one-time unforgeability security experiment the adversary obtains a single signature σ for a chosen message m and at the end it returns a forged signature σ^* on some message m^*. Note that the security experiment requires that m and m^* differ in at least one bit. The reduction guesses the bit position i^* of the message m and the value b^* of that bit in message m. If the reduction guesses the bit correctly, it can simulate all messages which have that bit set perfectly. The reduction does this by simply following the protocol and setting the value of the verification key correctly at all positions except value vk_{i^*,b^*}. For this element the reduction inputs the challenge of the FAPI-2 instance $e(g_1, g_2)^a$. Thus, the reduction can easily extract the solution g_1^a from the forgery of the adversary.

E.3 Omitted Theorems and Proofs for urSIG

Lemma 7 (Correctness of urSIG[rPKE, uSIG]). *If* rPKE *is correct and* uSIG *is correct then construction* urSIG[rPKE, uSIG] *in Fig. 7 is correct.*

Proof. Let (sk, vk) be in the support of urSIG.gen, message $m \in \mathcal{M}$ and $r \in \mathcal{R}$ and $\sigma \leftarrow \mathsf{urSIG}(\mathrm{sk}, m)$.

By definition of urSIG.sig,

$$\sigma = (\sigma_r, \sigma_x) = \left(\prod_{i=0}^{\ell-1} g_1^{r_{i,m_i}}, \prod_{i=0}^{\ell-1} g_1^{r_{i,m_i}\mathrm{dk}} H^{x_{i,m_i}}\right) \ .$$

Adversary $\mathcal{B}(g_1, g_2, A_T := e(g_1, g_2)^a)$

$b^* \xleftarrow{\$} [\ell], i^* \xleftarrow{\$} \{0,1\}$
For all $(i,b) \in ([\ell] \times \{0,1\}) \setminus \{i^*, b^*\}$:
  $z_{i,b} \xleftarrow{\$} \mathbb{Z}_p$
  $\mathrm{vk}_{i,b} \leftarrow e(g_1^{z_{i,b}}, g_2)$
$\mathrm{vk}_{i^*,b^*} \leftarrow A_T$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\texttt{Sign}(\cdot)}(\mathrm{vk})$
If $m^*_{i^*} = b^*$:
  Return $g_1^a \leftarrow \sigma^* \cdot \Pi_{i\in[\ell]\setminus\{i^*\}} g_1^{z_{i,m^*_i}}$
Abort

Oracle $\texttt{Sign}(m)$

If $m_{i^*} \neq b^*$:
  Return $\sigma \leftarrow \Pi_{i\in[\ell]} g_1^{z_{i,m_i}}$
Abort

Fig. 18. Adversary $\mathcal{B}$ against the FAPI-2 assumption.

By correctness of rPKE,

$$\mathsf{urSIG.vfy}(\mathrm{vk}, m, \sigma) = \mathsf{uSIG.vfy}(\mathrm{vk}, m, \prod_{i=0}^{\ell-1} H^{x_{i,m_i}}) \ .$$

Algorithm uSIG.vfy tests whether $e(\sigma, g_2) \stackrel{?}{=} \prod_{i=0}^{\ell-1} \mathrm{vk}'_{i,m_i}$. We continue,

$$e(\sigma, g_2) = e(\prod_{i=0}^{\ell-1} H^{x_{i,m_i}}, g_2) = e(\prod_{i=0}^{\ell-1} g_1^{x_{i,m_i}}, H) = \prod_{i=0}^{\ell-1} \mathrm{vk}'_{i,m_i} \ .$$

Clearly, by the group structure of the secret/verification key space, $(\mathsf{urSIG.nextSk}(\mathrm{sk}, r), \mathsf{urSIG.nextVk}(\mathrm{vk}, r)) \in \mathsf{urSIG.gen}$.

That for all $(\mathrm{vk}, \mathrm{sk}) \in \mathsf{urSIG.gen}$, $m \in \mathcal{M}$, an arbitrary number of randomizations resulting in a randomized secret key $\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$, a signature $\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, m)$ still verifies correctly follows directly from the correctness of rPKE.

Lemma 8 (IND-R **security of** urSIG)**.** *Clearly, for any adversary $\mathcal{A}$ against indistinguishability of randomizations of* urSIG*, there exists an adversary $\mathcal{B}$ against indistinguishability of randomizations of* rPKE *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{A},\mathsf{urSIG}} \leq \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{B},\mathsf{rPKE}} \ .$$

Lemma 9 (OT-EUF-CMA **security of** urSIG[rPKE, uSIG])**.** *Let* rPKE *be correct and homomorphic. For any adversary $\mathcal{A}$ against* OT-EUF-CMA *security of* urSIG[rPKE, uSIG]*, there exists an adversary $\mathcal{B}$ against* OT-EUF-CMA *of* uSIG *such that*

$$\mathsf{Adv}^{\mathsf{OT\text{-}EUF\text{-}CMA}}_{\mathcal{A},\mathsf{urSIG}} \leq \mathsf{Adv}^{\mathsf{OT\text{-}EUF\text{-}CMA}}_{\mathcal{B},\mathsf{uSIG}} \ .$$

Proof. Let $\mathcal{A}$ be an adversary in the $\mathsf{EUF\text{-}CMA}_{\mathsf{urSIG}}$ security experiment. We show how to construct an adversary $\mathcal{B}$ against $\mathsf{EUF\text{-}CMA}_{\mathsf{uSIG}}$, which uses $\mathcal{A}$ as a subroutine.

To simulate the verification key, adversary $\mathcal{B}$ samples a fresh rPKE key pair and forwards its own verification key input vk^* appended with dk to $\mathcal{A}$.

To simulate the single query $\texttt{Sign}(m)$ in the $\mathsf{EUF\text{-}CMA}_{\mathsf{urSIG}}$ security experiment to $\mathcal{A}$, adversary $\mathcal{B}$ calls the sign oracle provided by $\mathsf{EUF\text{-}CMA}_{\mathsf{uSIG}}$ on m and encrypts the returned signature with the rPKE scheme. To show that this is a valid simulation of a urSIG signature we argue as follows. The signature returned by the $\mathsf{EUF\text{-}CMA}_{\mathsf{uSIG}}$ experiment is of form $\sigma' := g^{\sum x_{i,h_i}}$, where $\{x\}_{i,h_i}$ are some of the dlogs of the secret key and $\{h\}_i$ are the bits of the hashed message. Encrypting σ' yields, $\mathsf{rPKE.enc}(\mathrm{ek}, \sigma') = (\prod g^{r_{i,h_i}}, \prod g^{r_{i,h_i}\mathrm{dk}} g^{x_{i,h_i}})$, which is a valid signature (σ_r, σ_x) in the $\mathsf{EUF\text{-}CMA}_{\mathsf{urSIG}}$ security experiment under vk.

To argue that the extraction of adversary $\mathcal{B}$ yields a valid forgery in the $\mathsf{EUF\text{-}CMA}_{\mathsf{uSIG}}$ security experiment we argue as follows. Adversary $\mathcal{A}$ returns a message signature pair $(m, (\prod g^{r_{i,h'_i}}, \prod g^{r_{i,h'_i}\mathrm{dk}} g^{x_{i,h'_i}})$, where h' is the hash of m. Thus $\mathsf{rPKE.dec}(\mathrm{dk}, (\prod g^{r_{i,h'_i}}, \prod g^{r_{i,h'_i}\mathrm{dk}} g^{x_{i,h'_i}})) = g^{\sum x_{i,h'_i}}$, which is a valid forgery in the $\mathsf{EUF\text{-}CMA}_{\mathsf{uSIG}}$ security experiment.

Corollary 1. *Since* uSIG *is verification key randomization smooth, so is* urSIG.

Adversary $\mathcal{B}^{\mathsf{Sign}(\cdot)}(\mathrm{vk}^*)$

$(\mathrm{ek}, \mathrm{dk}) \leftarrow \mathsf{rPKE.gen}()$
$\mathrm{vk} \leftarrow (\mathrm{vk}^*, \mathrm{dk})$
$(m, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{Sign}(\cdot)}(\mathrm{vk})$
$\sigma \leftarrow \mathsf{rPKE.dec}(\mathrm{dk}, \sigma^*)$
Return (m, σ)

Oracle $\mathtt{Sign}(m)$

$\sigma' \leftarrow \mathsf{Sign}(m)$
$\sigma \leftarrow \mathsf{rPKE.enc}(\mathrm{ek}, \sigma')$
Return σ

Fig. 19. Adversary $\mathcal{B}$ against OT-EUF-CMA of uSIG.

Lemma 10. *Signature scheme* urSIG *is* $\frac{1}{2^\lambda}$ *unverifiable under random verification keys.*

Proof. Let $(\mathrm{vk}, \mathrm{sk})$ by any key pair in the support of urSIG.gen and m and message in $\{0,1\}^*$. To estimate the probability that a random signature verifies under a random verification key we bound the following probability.

The signature verification algorithm checks whether $e(g, \sigma) = \prod_{i=0}^{\ell-1} \mathrm{vk}'_{i,h_i}$. Since vk' is an array of group elements from the target group $\mathbb{G}_T$ of the pairing, $\prod_{i=0}^{\ell-1} \mathrm{vk}'_{i,h_i}$ takes the value of a random element from $\mathbb{G}_T$. Thus,

$$\Pr_{\substack{\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk},\, m) \\ (\mathrm{vk}',\, \_) \xleftarrow{\$} \mathsf{urSIG.gen}}} \left[\mathsf{urSIG.vfy}(\mathrm{vk}', m, \sigma) = 1\right] = \Pr_{\substack{\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk},\, m) \\ t \xleftarrow{\$} \mathbb{G}_T}} \left[e(\sigma, g_2) = t\right] \leq \frac{1}{2^\lambda} .$$

E.4 Final Transformation to Strong Unforgeability of urSIG

To achieve strong unforgeability of urSIG we use the CHK transformation [MRY04, CHK04] using a strongly unforgeable signature. The full transformation is given in Fig. 20.

Lemma 11 (Strong Unforgeability [HWZ07]). *If* urSIG' *is* OT-EUF-CMA *and* SIG *is* SUF-CMA *then transformation* $\mathsf{urSIG}[\mathsf{urSIG}', \mathsf{SIG}]$ *given in Fig. 20 is* OT-SUF-CMA.

Proc urSIG.gen

$(\mathrm{sk}, \mathrm{vk}) \leftarrow \mathsf{urSIG}'.\mathsf{gen}()$

Proc $\mathsf{urSIG.rr}(\mathrm{sk})$

Return $\mathrm{sk} \leftarrow \mathsf{urSIG}'.\mathsf{rr}(\mathrm{sk})$

Proc $\mathsf{urSIG.nextSk}(\mathrm{sk}, r)$

$\mathrm{sk} \leftarrow \mathsf{urSIG}'.\mathsf{nextSk}(\mathrm{sk}, r)$

Proc $\mathsf{urSIG.nextVk}(\mathrm{vk}, r)$

$\mathrm{sk} \leftarrow \mathsf{urSIG}'.\mathsf{nextVk}(\mathrm{vk}, r)$

Proc $\mathsf{urSIG.sig}(\mathrm{sk}, m)$

$(\bar{\mathrm{sk}}, \bar{\mathrm{vk}}) \leftarrow \mathsf{SIG.gen}()$
$\sigma_1 \leftarrow \mathsf{urSIG}'.\mathsf{sig}(\mathrm{sk}, m = \bar{\mathrm{vk}})$
$\sigma_2 \leftarrow \mathsf{SIG.sig}(\bar{\mathrm{sk}}, m || \sigma_1)$
$\sigma \leftarrow (\sigma_1, \sigma_2, \bar{\mathrm{vk}})$

Proc $\mathsf{urSIG.vfy}(\mathrm{vk}, m, \sigma)$

Return $\mathsf{urSIG}'.\mathsf{vfy}(\mathrm{vk}, m = \bar{\mathrm{vk}}, \sigma_1)$
$\quad \wedge \mathsf{SIG.vfy}(\bar{\mathrm{vk}}, m || \sigma_1, \sigma_2)$

Fig. 20. Transformation of an OT-EUF-CMA secure, one-time signature scheme $\mathsf{urSIG}' = (\mathsf{urSIG}'.\mathsf{gen}, \mathsf{urSIG}'.\mathsf{sig}, \mathsf{urSIG}'.\mathsf{vfy}, \mathsf{urSIG}'.\mathsf{rr}, \mathsf{urSIG}'.\mathsf{nextSk}, \mathsf{urSIG}'.\mathsf{nextVk})$ and a OT-SUF-CMA secure one-time signature scheme $\mathsf{SIG} = (\mathsf{SIG.gen}, \mathsf{SIG.sig}, \mathsf{SIG.vfy})$ to an OT-SUF-CMA secure, updatable and randomizable one-time signature scheme $\mathsf{urSIG} = (\mathsf{urSIG}'.\mathsf{gen}, \mathsf{urSIG.sig}, \mathsf{urSIG.vfy}, \mathsf{urSIG}'.\mathsf{rr}, \mathsf{urSIG}'.\mathsf{nextSk}, \mathsf{urSIG}'.\mathsf{nextVk})$.

Corollary 2. *Since the transformation given in Fig. 20 only appends values to the signature and does not change anything else, all other security properties a* urSIG *scheme fulfills, still apply.*

F Proofs for our RKE Scheme

Here we finally give the full proofs for Robustness, Recover security, Authenticity, Key Indistinguishability and Anonymity of RKE[urPKE, urSIG, H, PRG].

F.1 Robustness

Lemma 12. $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$ *is robust.*

Proof. The output (stS, c) of $\mathsf{RKE.snd}$ will never be $\perp$ by definition of the syntax of urSIG and urPKE. To show that $\mathrm{stR} = \mathrm{stR}'$ if $k_R = \perp$ we continue as follows. In the beginning of a call to $\mathsf{RKE.rcv}$ the key k is set to $\perp$. Only if the one-time signature embedded in the encapsulation does not verify then the key stays $\perp$. If the signature does not verify, then also the receiver state does not change.

F.2 Recover Security

Theorem 4 (Recover Security of $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$). *Let* urPKE *be a correct, updatable and randomizable public key encryption scheme,* urSIG *a correct, updatable and randomizable one-time signature and* PRG *a pseudorandom generator. Let* $\mathsf{H}\colon \{0,1\}^* \mapsto 2^\lambda \times \mathcal{R}_{\mathsf{urPKE}} \times \mathcal{R}_{\mathsf{urSIG}}$ *be a random oracle, where* $\mathcal{R}_{\mathsf{urPKE}}$ *and* $\mathcal{R}_{\mathsf{urSIG}}$ *are the randomization spaces of* urPKE *and* urSIG*, respectively. We show that if* H *is* CR *then* $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$ *is* $(q_H, \frac{{q_H}^2}{2^\lambda})$-$\mathsf{RECOV}$*, where* q_H *is the number of calls to the random oracle.*

Proof. Consider the sequence of games in Fig. 21.

Game $\mathsf{RECOV}_{\mathsf{RKE}}(\mathcal{A}) = \mathsf{G}_0, \mathsf{G}_1$
$\boldsymbol{cadk} \leftarrow [\cdot]$
$(s, r) \leftarrow (0, 0)$
$\mathrm{is} \leftarrow \mathtt{tru}$
$(\mathrm{stS}, \mathrm{stR}) \xleftarrow{\$} \mathsf{RKE.init}$
Invoke $\mathcal{A}$
Stop with 0

Oracle $\mathtt{Snd}(\mathrm{ad})$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}_{\mathrm{next}}, \mathrm{sk}_{\mathrm{next}}) \leftarrow \mathsf{urSIG.gen}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
$\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r_{\mathsf{urPKE}})$
$\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
$\boldsymbol{cadk}[s] \leftarrow (c, \mathrm{ad}, k)$
$s \leftarrow s + 1$
Return (c, k)

Oracle $\mathsf{H}(x)$
If $h[x] \neq \perp$:
  Return $h[x]$
$h[x] \xleftarrow{\$} \mathbb{G} \times \mathcal{K}_{\mathsf{urPKE}} \times \mathcal{K}_{\mathsf{urSIG}}$
If $\exists x' \neq x$ s.t. $h[x] = h[x']$:
  ABORT // G_1
Return $h[x]$

Oracle $\mathtt{Expose}$
Return $(\mathrm{stS}, \mathrm{stR})$

Oracle $\mathtt{Rcv}(c, \mathrm{ad})$
$k \leftarrow \perp$
$(\mathrm{dk}, \mathrm{vk}) \leftarrow \mathrm{stR}$
$(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
$(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
Require $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \perp$
If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
  $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
  $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
  $\mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r_{\mathsf{urPKE}})$
  $\mathrm{stR} \leftarrow (\mathrm{dk}, \mathrm{vk})$
If $k = \perp$:
  Return
$(c', \mathrm{ad}', k_S) \leftarrow \boldsymbol{cadk}[r]$
If $(c, \mathrm{ad}) \neq (c', \mathrm{ad}')$:
  $\mathrm{is} \leftarrow \mathtt{fal}$
If $\mathrm{is} = \mathtt{fal} \wedge \exists s^* : (\_, c, \_) = \boldsymbol{cadk}[s^*]$:
  Stop with 1
If $(c, \mathrm{ad}) = (c', \mathrm{ad}')$:
  $r \leftarrow r + 1$
Return

Oracle $\mathtt{RR}$
$\mathrm{stS} \xleftarrow{\$} \mathsf{RKE.rr}(\mathrm{stS})$

Fig. 21. Games for the proof of Theorem 4.

Experiment Exp_0. This game is equivalent to $\mathsf{RECOV}_{\mathsf{RKE}}$, where H is modeled as a standard lazy sampled random oracle.

Experiment Exp_1. In this game we exclude that the adversary can predict the output of the random oracle. To this end the random oracle aborts if queried on the same value twice. To mitigate trivial aborts of the game when the game itself should call the random oracle multiple times on the same value we replace all but the first call on a distinct value with an access to the data structure underlying the random oracle holding the lazy sampled values.

Since the outputs of the random oracle are uniformly at random, we can bound the probability of an adversary distinguishing both games as,

$$\left|\Pr\left[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right]\right| \le \frac{q_S + q_H}{2^\lambda} \ .$$

Experiment Exp_2. In this game we exclude collisions in the random oracle. To this end the random oracle aborts if queried on two distinct outputs which would result in the same output.

Since the outputs of the random oracle are uniformly at random, we can bound the probability of an adversary distinguishing both games as,

$$\left|\Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1\right]\right| \le \frac{(q_S + q_{CS})^2}{2^\lambda} \ .$$

We now argue that the adversary cannot break $\mathsf{RECOV}_{\mathsf{RKE}}$ security.

By the definition of $\mathsf{RECOV}_{\mathsf{RKE}}$ security, there exists some (minimum) sender stage $s = i$ such that $\boldsymbol{cadk}[s] \leftarrow (c, \mathrm{ad}, k)$, and a receiver stage $r = i$ such that $\mathtt{Rcv}(c', \mathrm{ad}')$ is called and $(c, \mathrm{ad}) \neq (c', \mathrm{ad}')$. For the adversary to win there must exist some sender stage $s = s^*$ such that $(c^*, \_, \_) = \boldsymbol{cadk}[s^*]$ and if $\mathtt{Rcv}(c^*, \mathrm{ad}^*)$ is called when receiver stage $r = r^*$ such that $r^* > i$, then the receiver computes $k \neq \bot$.

Note that in stage $s = i$ the sender computes $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$, and the receiver computes $(k', r'_{\mathsf{urPKE}}, r'_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k'_{\mathsf{H}}, c', \mathrm{ad}')$. Since, by the previous game, we abort when collisions occur in the random oracle and H is modelled as a random oracle, it follows that if $(c, \mathrm{ad}) \neq (c', \mathrm{ad}')$, then $r_{\mathsf{urSIG}} \neq r'_{\mathsf{urSIG}}$, and are random and independent values. Similarly, $r_{\mathsf{urPKE}} \neq r'_{\mathsf{urPKE}}$ are random and independent values.

As proven earlier, urSIG signatures do not verify under uniformly random keys. The sender computes the next signing key $\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$, and the receiver computes the next verification key as $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r'_{\mathsf{urSIG}})$. Since $r_{\mathsf{urSIG}}, r'_{\mathsf{urSIG}}$ are both uniformly random and independent values, vk is now independent of sk, and thus the receiver in stage $r = i + 1$ will not verify any signature created by the sender in stage $s = i + 1$. Since the signature accepted by the receiver in stage $r = i + 1$ differs from the signature created by the sender in stage $i + 1$, then the ciphertext created by the sender in stage $s = i + 1$ differs from the ciphertext received by the receiver in stage $r = i + 1$ (since, if $\sigma \neq \sigma'$, but $\sigma \oplus \mathsf{PRG}(k_S) = \sigma' \oplus \mathsf{PRG}(k'_S)$, then $k_S \neq k'_S$ and finally $c_{\mathsf{urPKE}} = \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}})) \neq c'_{\mathsf{urPKE}}$). Applying the same argument, it follows that the sender's computed sk in stage $s = i + 1$ is independent of the vk generated by the receiver in stage $r = i$, and it follows that no sender ciphertext generated in stage $s = i' \neq i$ will verify in receiver stage $r = i$.

We now demonstrate that the adversary cannot cause the verification key vk used by the receiver in stage $r = i + 1$ to be set to a previous receiver verification key vk. If so, then by applying the same arguments as before, it follows that the sender's signatures in all previous stages $s = i'' < i$ cannot verify in stage $r = i + 1$, and thus the receiver will never output $k \neq \bot$ when receiving (c, ad) where c was output by the sender. The proof is straightforward: to cause a collision on some previously computed vk, the adversary must generate a k_{H} value such that $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$ where $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$. Thus, the adversary must have previously queried k_{H} to the random oracle, which we exclude in Game 1. Thus, the adversary cannot generate such a k_{H} value.

Thus, the adversary cannot cause the receiver to accept in stage $r = i + 1$ by replaying an old ciphertext output by the sender, nor will the sender ever generate a new signature that will verify under the verification key vk used in stage $r = i + 1$. We note that these arguments apply to all stage $r = i^* > i$, and thus in Game 2 we have that the adversary cannot break $\mathsf{RECOV}_{\mathsf{RKE}}$ security.

F.3 Authenticity

Intuition. In this proof we reduce the AUTH security of RKE to the $\mathsf{SUF\text{-}CMA}$ security of urSIG. Since our construction samples a new urSIG instance at every call to oracle $\mathtt{Snd}$, we start by doing a standard multi instance to single instance reduction. At a randomly selected call to oracle `Snd` we simulate the underlying urSIG instance with the oracles of the SUF-CMA security experiment. If the adversary breaks AUTH at exactly that simulated urSIG session the reduction can extract from that a solution in the SUF-CMA security experiment of urSIG.

Theorem 5 (Authenticity of RKE[urPKE, urSIG, H, PRG]**).** *Let* urPKE *be a correct updatable and randomizable public key encryption scheme,* urSIG *a correct, updatable and randomizable one-time signature and* PRG *a pseudorandom generator. Let* $\mathsf{H}\colon \{0,1\}^* \mapsto 2^\lambda \times \mathcal{R}_{\mathsf{urPKE}} \times \mathcal{R}_{\mathsf{urSIG}}$ *be a programmable random oracle, where* $\mathcal{R}_{\mathsf{urPKE}}$ *and* $\mathcal{R}_{\mathsf{urSIG}}$ *are the randomization spaces of* urPKE *and* urSIG*, respectively. If* urSIG *is* OT-SUF-CMA *secure,* H *is* OW*, and* RKE[urPKE, urSIG, H, PRG] *is* $(q_S + q_H, \varepsilon)$-RECOV *we show that* RKE[urPKE, urSIG, H, PRG] *is* AUTH*, with*

$$\mathsf{Adv}^{\mathsf{AUTH}}_{\mathsf{RKE}} \leq q_S \cdot \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{urSIG}} + \frac{1}{2^\lambda} \ ,$$

where q_H *and* q_S *are the number of calls to the random oracle and oracle* `Sign`*, respectively.*

Intuition. In this proof we reduce the AUTH security of RKE to the SUF-CMA security of urSIG. Since our construction samples a new urSIG instance at every call to oracle `Snd`, we start by doing a standard multi instance to single instance reduction. At a randomly selected call to oracle `Snd` we simulate the underlying urSIG instance with the oracles of the SUF-CMA security experiment. If the adversary breaks AUTH at exactly that simulated urSIG session the reduction can extract from that a solution in the SUF-CMA security experiment of urSIG.

Proof. Let AUTH′ be the same security experiment as AUTH, where the only difference is that AUTH′ aborts if the adversary is able to predict the output of the random oracle. This happens with probability at most $\frac{1}{2^\lambda}$.

Let $\mathcal{A}$ be an adversary in the AUTH′ experiment of RKE. We show how to construct an adversary $\mathcal{B}$ against the SUF-CMA security of urSIG. To embed the urSIG instance, adversary $\mathcal{B}$ embeds the instance at an index uniformly at random in the number of queries to oracle `Snd`. Thus adversary $\mathcal{B}$ samples an index $i^* \xleftarrow{\$} [q_S]$ and samples for all calls to `Snd` the states according to the construction. The signature in the i^*th call to oracle `Snd` however will be replaced by the output of the Sign oracle adversary $\mathcal{B}$ is supplied with by the SUF-CMA security experiment.

For the simulation to be well-distributed the signature embedded in the i^*th call to `Snd` must verify under the current verification key vk in the receivers state. We proceed by computing the randomness r_{urSIG} such that $\mathsf{urSIG.nextVk}(\mathrm{vk}, r_{\mathsf{urSIG}})$ outputs vk^* and program the random oracle to output that r_{urSIG}. By the verification key randomization smoothness there exists an algorithm findUpdate which on input vk^* outputs well distributed r_{urSIG} and vk s.t. $\mathrm{vk}^* = \mathsf{urSIG.nextVk}(\mathrm{vk}, r_{\mathsf{urSIG}})$. We program the random oracle on $k_{\mathsf{H}}, c, \mathrm{ad}$ to output $(\_, r_{\mathsf{urSIG}}, \_)$. Since vk′ is uniformly at random, r_{urSIG} and vk are well-distributed.

If adversary $\mathcal{A}$ produces a forgery while vk^* is part of stR then we can extract easily.

F.4 Key Indistinguishability

Theorem 6 (Key Indistinguishabilty of RKE[urPKE, urSIG, H, PRG]**).** *Let* urPKE *be a correct updatable and randomizable public key encryption scheme,* urSIG *a correct updatable and randomizable one-time signature and* PRG *a pseudorandom generator. Let* $\mathsf{H}\colon \{0,1\}^* \mapsto 2^\lambda \times \mathcal{R}_{\mathsf{urPKE}} \times \mathcal{R}_{\mathsf{urSIG}}$ *be a programmable random oracle, where* $\mathcal{R}_{\mathsf{urPKE}}$ *and* $\mathcal{R}_{\mathsf{urSIG}}$ *are the randomization spaces of* urPKE *and* urSIG*, respectively. For any adversary* $\mathcal{A}$ *against* KIND *security of* RKE[urPKE, urSIG, H, PRG]*, there exists an adversary* $\mathcal{B}$ *against* IND-C *security of* urPKE *and an adversary* $\mathcal{C}$ *against* PRG *such that*

$$\mathsf{Adv}^{\mathsf{KIND}}_{\mathcal{A},\mathsf{RKE}} \leq (q_S + q_{CS}) \left(\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{B},\mathsf{urPKE}} + \mathsf{Adv}_{\mathcal{C},\mathsf{PRG}}\right) + \frac{1}{2^\lambda} \ .$$

Proof. Consider the sequence of games in Fig. 23.

Experiment Exp_0. This game is the same game as $\mathsf{IND\text{-}C}^b_{\mathsf{RKE}}$, where oracle `Rcv` is changed only notationally. Since $k \neq \perp$ in $\mathsf{IND\text{-}C}^b_{\mathsf{RKE}}$ will only be true if the one time signature verifies, the adversary can not distinguish the rewriting of `Rcv`.

```
Adversary B^{Sign(·),Up}(vk*)
i* ←$ [q_S]
cad := [·], xS := ∅
(s, r) ← (0, 0)
(ek, dk) ←$ urPKE.gen
(vk, sk) ←$ urSIG.gen
stS ← (ek, sk)
stR ← (dk, vk)
If i* = 0:
    stR ← (dk, vk*)
Invoke A
Stop with 0

Oracle RR
(ek, sk) ← stS
ek ←$ urPKE.rr(ek)
If r = i*:
    Up
Else :
    sk ←$ urSIG.rr(sk)
stS ← (ek, sk)
Return

Oracle Expose_S
xS ←∪ {s}
Return stS
```

```
Oracle Snd(ad)
(ek, sk) ← stS
sk' ← sk
k_H, k_S ←$ K
(vk', sk') ← urSIG.gen
If s = i*:
    (R, vk') ← findUpdate(vk*)
    Program H to R on (k_H, c, ad)
c_urPKE ←$ urPKE.enc(ek, (k_H, k_S, vk'))
If r = i*:
    σ ← Sign(c_urPKE, ad)
Else
    σ ←$ urSIG.sig(sk, (c_urPKE, ad))
c ← (c_urPKE, σ ⊕ PRG(k_S))
(k, r_urPKE, r_urSIG) ← H(k_H, c, ad)
sk ← urSIG.nextSk(sk', r_urSIG)
ek ← urPKE.nextEk(ek, r_urPKE)
stS ← (ek, sk)
cad ←∪ (c, ad, sk')
Return (c, k)

Oracle Expose_R
Return stR
```

```
Oracle Rcv(c, ad)
k ← ⊥
(dk, vk) ← stR
(c_urPKE, σ') ← c
(k_H, k_S, vk') ← urPKE.dec(dk, c_urPKE)
Require (k_H, k_S, vk') ≠ ⊥
If urSIG.vfy(vk, (c_urPKE, ad), σ' ⊕ PRG(k_S))
    (k, r_urPKE, r_urSIG) ← H(k_H, c, ad)
    vk ← urSIG.nextVk(vk', r_urSIG)
    dk ← urPKE.nextDk(dk, r_urPKE)
    stR ← (dk, vk)
    If is = tru ∧ (c, ad) ≠ cad[r]:
        is ← fal
        If r ∉ xS: Stop with 1
    r ← r + 1
Return

Random Oracle H(x)
If h[x] ≠ ⊥:
    Return h[x]
h[x] ←$ H
Return h[x]
```

Fig. 22. Adversary $\mathcal{B}$ against SUF-CMA for the proof of Theorem 5.

Experiment Exp_1. In this game we replace the outputs of random oracle H by uniform random. Since k_h is random, the output of hash function in RKE.snd is unpredictable. So,

$$\left|\Pr\left[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \frac{1}{2^\lambda} \ .$$

Experiment Exp_2. In this game we replace the randomized instances of the RKE states with independently sampled instances. By $q_S + q_{CS}$ times application of Instance independence of urPKE, this is undetectable for the adversary. Since r_{urSIG} is uniformly at random, all independently sampled urSIG instances stay independent. Since Instance independence holds statistically,

$$\Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1\right] \ .$$

Note that since the individual RKE instances are independent, any corruption of a receiver state does not reveal anything about prior receiver states and any sender state corruption does not reveal anything about prior and future sender states.

Experiment Exp_3. in this game we replace the encapsulation output of urPKE with uniform randomness and subsequently the whole ciphertext c with uniform randomness. To show that no adversary is able to detect this we distinguish the behavior of the adversary. If the adversary queries at any point oracle $\mathtt{Expose}_R$ then the adversary can not distinguish statistically. If at query time of $\mathtt{Expose}_R$ the in sync variable is true then all challenge ciphertexts must be received. Since after a successful receive the game receiver state is statistically independent of prior states, the adversary learns only a state which can not decrypt any prior challenge ciphertexts. Since any call to $\mathtt{Expose}_R$ while is $=$ `tru` sets the exposed receiver variable xR to be always true, there can no be future calls to oracle `ChallSnd`. So all future outputs of the game are statistically independent of bit b. If at query time of $\mathtt{Expose}_R$ the in sync variable is set to false then we further distingusih the relative order of impersonation an call to oracle `ChallSnd`. If the adversary i) impersonated, ii) called oracle `ChallSnd` $\to c$, iii) called oracle $\mathtt{Expose}_R \to$ stR then by instance independence stR is statistically independent from the game's sender state at time ii). Thus the output of oracle $\mathtt{Expose}_R$ can not be used to decrypt c. If the adversary i) called oracle `ChallSnd` $\to c$, ii) impersonated, iii) called oracle $\mathtt{Expose}_R \to$ stR then by instance independence stR is statistically

Game $\mathsf{KIND}^b_{\mathsf{RKE}}(\mathcal{A}) = \mathsf{G}_0, \mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3$

· $\boldsymbol{I} \leftarrow [\cdot]$
· For $i \in [q_S + q_{CS}]$:
· $\boldsymbol{I}.\mathsf{append}(\mathsf{RKE.init}())$
· $(\mathrm{stS}, \mathrm{stR}) \leftarrow \boldsymbol{I}[s]$
· $\boldsymbol{cad} \leftarrow [\cdot]$
· $(\boldsymbol{cc}, \boldsymbol{rcvd}) \leftarrow (\emptyset, \emptyset)$
· $\mathrm{is} \leftarrow \mathtt{tru}$
· $\mathrm{xR} \leftarrow \mathtt{fal}$
· $(s, r) \leftarrow (0, 0)$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle $\mathtt{Snd}(\mathrm{ad})$

$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{urSIG.gen}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}'))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$(\mathrm{stS}', (\mathrm{sk}', \mathrm{vk}')) \leftarrow \boldsymbol{I}[s]$ // G_2
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \xleftarrow{\$} (\mathbb{G} \times \mathcal{K} \times \mathcal{K})$ // G_1
$\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}', r_{\mathsf{urSIG}})$
$\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r_{\mathsf{urPKE}})$
$\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
· $\boldsymbol{cad}[s] \leftarrow (c, \mathrm{ad})$
· $s \leftarrow s + 1$
Return (c, k)

Oracle $\mathtt{RR}$

$\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$
Return

Oracle $\mathtt{Expose}_S$

Return stS

Oracle $\mathtt{ChallSnd}(\mathrm{ad})$

· Require $\mathrm{xR} \neq \mathtt{tru}$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{urSIG.gen}$
$(\mathrm{stS}', (\mathrm{sk}', \mathrm{vk}')) \leftarrow \boldsymbol{I}[s]$ // G_2
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}'))$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathbb{G}^2$ // G_3
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$c \leftarrow (c_{\mathsf{urPKE}}, \xleftarrow{\$} \mathcal{S})$ // G_3
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \xleftarrow{\$} (\mathbb{G} \times \mathcal{K} \times \mathcal{K})$ // G_1
$\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}', r_{\mathsf{urSIG}})$
$\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r_{\mathsf{urPKE}})$
$\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
· $\boldsymbol{cad}[s] \leftarrow (c, \mathrm{ad})$
· $\boldsymbol{cc} \xleftarrow{\cup} \{s\}$
· $s \leftarrow s + 1$
If $b = 1$: $k \xleftarrow{\$} \mathcal{K}$
Return (c, k)

Oracle $\mathtt{Rcv}(c, \mathrm{ad})$

$k \leftarrow \bot$
$(\mathrm{dk}, \mathrm{vk}) \leftarrow \mathrm{stR}$
$(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
$(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
Require $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \bot$
If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
    $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
    $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
    $\mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r_{\mathsf{urPKE}})$
    $\mathrm{stR} \leftarrow (\mathrm{dk}, \mathrm{vk})$
·   If $(c, \mathrm{ad}) \neq \boldsymbol{cad}[r]$:
· $\mathrm{is} \leftarrow \mathtt{fal}$
·   If $(c, \mathrm{ad}) = \boldsymbol{cad}[r]$:
· $\boldsymbol{rcvd} \xleftarrow{\cup} \{r\}$
· $r \leftarrow r + 1$
If $\mathrm{is} = \mathtt{tru}$:
        $(\_, \mathrm{stR}) \leftarrow \boldsymbol{I}[r]$ // G_2
Return

Oracle $\mathtt{Expose}_R$

· If $\mathrm{is} = \mathtt{tru}$:
·   Require $\boldsymbol{cc} \subseteq \boldsymbol{rcvd}$
· $\mathrm{xR} \leftarrow \mathtt{tru}$
Return stR

Fig. 23. Games for the proof of Theorem 6.

independent from the game's sender state at time i). Thus the output of oracle $\mathtt{Expose}_R$ can not be used to decrypt c.

If the adversary does not call oracle $\mathtt{Expose}_R$ at all then we show in Lemma 13 how the hardness of distinguishing between G_2 and G_3 is bounded by $\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{B},\mathsf{urPKE}} + \mathsf{Adv}_{\mathcal{C},\mathsf{PRG}}$.

Lemma 13. *The advantage of any PPT adversary distinguishing between games* G_2 *and* G_3 *is bounded by* $(q_S + q_{CS}) \cdot \left(\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{B},\mathsf{urPKE}} + \mathsf{Adv}_{\mathcal{C},\mathsf{PRG}}\right)$.

Proof. Let $\mathcal{A}$ be an adversary distinguishing G_2 and G_3. Assume that $\mathcal{A}$ does not call oracle $\mathtt{Expose}_R$. We then show how to construct an adversay $\mathcal{B}$ against $\mathsf{IND\text{-}C}$ of urPKE.

To this end we give a hybrid argument where hybrids are defined over every independent RKE instance. So in Fig. 24 we introduce hybrids H_q, where $q \in [q_S + q_{CS}]$. We define the hybrids s.t. $\mathsf{H}_0 := \mathsf{G}_3$ and $\mathsf{H}_{q_S+q_{CS}} := \mathsf{G}_2$.

We show in Fig. 25 how to construct adversaries $\mathcal{B}_q$ which bound the advantage of any adversary distinguishing hybrids H_q and H_{q+1}, with the advantage of breaking $\mathsf{IND\text{-}C}_{\mathsf{urPKE}}$.

In the following we show that if adversary $\mathcal{B}_q$ is executed in the real or random version of $\mathsf{IND\text{-}C}_{\mathsf{urPKE}}$ then $\mathcal{B}_q$ simulates the hybrid H_q or H_{q+1}, respectively.

Hybrid H_q expects after the $q-1$th call to oracle `Snd` that the underlying sender state is independent from previous sender states. Thus adversary $\mathcal{B}_q$ can forward all oracles calls to the oracles provided by the $\mathsf{IND\text{-}C}_{\mathsf{urPKE}}$ security experiment up until adversary $\mathcal{A}$ queries `Snd` for the qth time.

Oracle $\mathsf{ChallSnd}$ provided by $\mathsf{IND\text{-}C}^b_{\mathsf{urPKE}}$ returns only if $b = 0$, ciphertexts which are a function of its underlying encryption key ek. Thus the output of `ChallSnd` is consistent with the outputs of oracles $\mathsf{RR}, \mathsf{Expose}_S$ and $\mathsf{ChallExpose}_S$.

Oracle `Rcv` behaves as follows. If the adversary calls oracle `Rcv` after the q-th successful receive of a ciphertext, adversary $\mathcal{B}$ forwards the underlying c_{urPKE} ciphertext to its decryption oracle only if is $=$ `tru`. Only if there were no impersonations in $\mathsf{KIND}_{\mathsf{RKE}}$, the q-th updated receiver state would match the q-th updated sender state. If there was an impersonation then oracle `Rcv` advances the receiver state according to the construction. If the adversary calls oracle `Rcv` on a challenge then oracle Dec returns the adversaries input to a call to oracle $\mathsf{ChallSnd}$. Thus the output of Dec is well-defined and the state progression of the receiver state is indistinguishable.

Finally, since k_S is now statistically independent from c_{urPKE}, by PRG security of PRG, the adversary can not distinguish.

In total,

$$\mathsf{Adv}^{\mathsf{KIND}}_{\mathcal{A},\mathsf{RKE}} \leq (q_S + q_{CS}) \left(\mathsf{Adv}^{\mathsf{IND\text{-}C}}_{\mathcal{B},\mathsf{urPKE}} + \mathsf{Adv}_{\mathcal{C},\mathsf{PRG}}\right) + \frac{1}{2^\lambda} \ .$$

F.5 Anonymity

Proof (Theorem 1). Consider the sequence of games in Fig. 26.

Experiment Exp_0. This game is equivalent to $\mathsf{ANON}^b_{\mathsf{RKE}}$.

Experiment Exp_1. In this game we replace the outputs of random oracle H by uniform random. Since k_h is random, the outputs of hash function in $\mathsf{RKE.snd}$ are random. So,

$$\left|\Pr\left[\mathsf{G}_0^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right]\right| \leq \frac{1}{2^\lambda} \ .$$

Experiment Exp_2. In this game we replace the randomized instances of the RKE states with independently sampled instances. By $q_S + q_{CS}$ times application of Instance independence of urPKE, this is undetectable for the adversary. Since r_{urSIG} is uniformly at random, all independently sampled urSIG instances stay independent. Since Instance independence holds statistically,

$$\Pr\left[\mathsf{G}_1^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1\right] \ .$$

Note that since the individual RKE instances are independent any corruption of a receiver state does not reveal anything about prior receiver states and any sender state corruption does not reveal anything about prior and future sender states.

Experiment Exp_3. This game always outputs the random utopian world output in oracle $\texttt{ChallExpose}_R$. We now argue that the adversary can not distinguish this game from the prior game. Therefore we first exclude the case that the adversary did not call $\texttt{ChallExpose}_R$. If so, the adversary clearly can not distinguish between G_2 and G_3. Further, if the adversary were to call $\texttt{ChallExpose}_R$ more than once then this would constitute a trivial attack.

Let the adversary w.l.o.g. call $\texttt{ChallExpose}_R$ on some instance. By definition of our requirements for trivial attacks there must be no unreceived ciphertexts output by either `Snd` or `ChallSnd`. Thus there is

Hybrids H_q

$\boldsymbol{I} \leftarrow [\cdot]$
For $i \in [q_S + q_{CS}]$:
  $\boldsymbol{I}.\mathsf{append}(\mathsf{RKE.init}())$
$i \leftarrow 0$
$(\mathrm{stS}, \mathrm{stR}) \leftarrow \boldsymbol{I}[s]$
$\boldsymbol{cad} \leftarrow [\cdot]$
$(\boldsymbol{cc}, \boldsymbol{rcvd}) \leftarrow (\emptyset, \emptyset)$
$\mathrm{is} \leftarrow \mathtt{tru}$
$\mathrm{xR} \leftarrow \mathtt{fal}$
$(s, r) \leftarrow (0, 0)$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle $\mathtt{Snd}(\mathrm{ad})$

$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{urSIG.gen}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}'))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$(\mathrm{stS}', (\mathrm{sk}', \mathrm{vk}')) \leftarrow \boldsymbol{I}[s]$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \xleftarrow{\$} (\mathbb{G} \times \mathcal{K} \times \mathcal{K})$
$\mathrm{stS} \leftarrow \mathrm{stS}'$
$\boldsymbol{cad}[s] \leftarrow (c, \mathrm{ad})$
$s \leftarrow s + 1$
Return (c, k)

Oracle $\mathsf{H}(x)$

If $r[x] \neq \bot$:
  Return $r[x]$
$r[x] \xleftarrow{\$} \mathbb{G} \times \mathcal{K} \times \mathcal{K}$
If $\exists x' \neq x$ s.t. $r[x] = r[x']$:
  ABORT
Return $r[x]$

Oracle $\mathtt{RR}$

$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
If $s \leq q$:
  $\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$
If $s \neq q$:
  $\mathrm{ek} \xleftarrow{\$} \mathcal{E}$
$\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$
$\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
Return stS
Return

Oracle $\mathtt{Expose}_S$

Return stS

Oracle $\mathtt{ChallSnd}(\mathrm{ad})$

Require $\mathrm{xR} \neq \mathtt{tru}$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{urSIG.gen}$
$(\mathrm{stS}', (\mathrm{sk}', \mathrm{vk}')) \leftarrow \boldsymbol{I}[s]$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}'))$
If $s > q$:
  $c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, \xleftarrow{\$} \mathcal{K})$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \xleftarrow{\$} (\mathbb{G} \times \mathcal{K} \times \mathcal{K})$
$\mathrm{stS} \leftarrow \mathrm{stS}'$
$\boldsymbol{cad}[s] \leftarrow (c, \mathrm{ad})$
$\boldsymbol{cc} \xleftarrow{\cup} \{s\}$
$s \leftarrow s + 1$
Return (c, k)

Oracle $\mathtt{Rcv}(c, \mathrm{ad})$

$k \leftarrow \bot$
$(\mathrm{dk}, \mathrm{vk}) \leftarrow \mathrm{stR}$
$(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
$(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
Require $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \bot$
If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
  $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
  $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
  $\mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r_{\mathsf{urPKE}})$
  $\mathrm{stR} \leftarrow (\mathrm{dk}, \mathrm{vk})$
  If $(c, \mathrm{ad}) \neq \boldsymbol{cad}[r]$:
    $\mathrm{is} \leftarrow \mathtt{fal}$
  If $(c, \mathrm{ad}) = \boldsymbol{cad}[r]$:
    $\boldsymbol{rcvd} \xleftarrow{\cup} \{r\}$
    $r \leftarrow r + 1$
If $\mathrm{is} = \mathtt{tru}$:
  $(\_, \mathrm{stR}) \leftarrow \boldsymbol{I}[r]$
Return

Oracle $\mathtt{Expose}_R$

If $\mathrm{is} = \mathtt{tru}$:
  Require $\boldsymbol{cc} \subseteq \boldsymbol{rcvd}$
  $\mathrm{xR} \leftarrow \mathtt{tru}$
Return stR

Fig. 24. Hybrids H_q for the proof of Lemma 13.

nothing besides impersonation ciphertexts or not authentic ciphertexts for the adversary to deliver to $\mathtt{Rcv}$. By definition of the requirements against matching trivial attacks, the adversary must not have corrupted the sender before calling oracle $\mathtt{ChallExpose}_R \to \mathrm{stR}$, such that the prior exposed sender state and stR match in the real world. Since the adversary does not know an exposed sender state which could have been used to impersonate the real utopian game, the real world utopian game can not be impersonated before and after calling oracle $\mathtt{ChallExpose}_R$.

We now distinguish the two cases whether the random world was impersonated. Assume the adversary did not attempt to impersonate the random world execution and calls oracle $\mathtt{ChallExpose}_R$. The game

Adversary $\mathcal{B}_q^{\mathsf{Snd},\mathsf{ChallSnd},\mathsf{RR},\mathsf{Dec}}(\mathrm{ek}^*)$

$\boldsymbol{I} \leftarrow [\cdot]$
For $i \in [q_S + q_{CS}]$:
  $\boldsymbol{I}.\mathsf{append}(\mathsf{RKE.init}())$
$(\mathrm{stS}, \mathrm{stR}) \leftarrow \boldsymbol{I}[s]$
$\boldsymbol{cad} \leftarrow [\cdot]$
$(\boldsymbol{cc}, \boldsymbol{rcvd}) \leftarrow (\emptyset, \emptyset)$
$\mathrm{is} \leftarrow \mathtt{tru}$
$\mathrm{xR} \leftarrow \mathtt{fal}$
$(s, r) \leftarrow (0, 0)$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle $\mathtt{Snd}(\mathrm{ad})$

$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{urSIG.gen}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}'))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$(\mathrm{stS}', (\mathrm{sk}', \mathrm{vk}')) \leftarrow \boldsymbol{I}[s]$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \xleftarrow{\$} (\mathbb{G} \times \mathcal{K} \times \mathcal{K})$
$\mathrm{stS} \leftarrow \mathrm{stS}'$
$\boldsymbol{cad}[s] \leftarrow (c, \mathrm{ad})$
$s \leftarrow s + 1$
Return (c, k)

Oracle $\mathtt{RR}$

If $s = q$:
  $\mathsf{RR}()$
Else:
  $\mathrm{stS} \leftarrow \mathsf{RKE.rr}(\mathrm{stS})$
Return

Oracle $\mathtt{Expose}_S$

If $s = q$:
  $(\_, \mathrm{sk}) \leftarrow \mathrm{stS}$
  return $(\mathrm{ek}^*, \mathrm{sk})$
Return stS

Oracle $\mathtt{ChallSnd}(\mathrm{ad})$

Require $\mathrm{xR} \neq \mathtt{tru}$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}', \mathrm{sk}') \leftarrow \mathsf{urSIG.gen}$
$(\mathrm{stS}', (\mathrm{sk}', \mathrm{vk}')) \leftarrow \boldsymbol{I}[s]$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}'))$
If $s = q$:
  $c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{ChallSnd}(k_{\mathsf{H}}, k_S, \mathrm{vk}')$
If $s > q$:
  $c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, \xleftarrow{\$} \mathcal{K})$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \xleftarrow{\$} (\mathbb{G} \times \mathcal{K} \times \mathcal{K})$
$\mathrm{stS} \leftarrow \mathrm{stS}'$
$\boldsymbol{cad}[s] \leftarrow (c, \mathrm{ad})$
$\boldsymbol{cc} \xleftarrow{\cup} \{s\}$
$s \leftarrow s + 1$
Return (c, k)

Oracle $\mathtt{Rcv}(c, \mathrm{ad})$

$k \leftarrow \bot$
$(\mathrm{dk}, \mathrm{vk}) \leftarrow \mathrm{stR}$
$(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
If $r = q \wedge \mathrm{is} = \mathtt{tru}$:
  $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{Dec}(c_{\mathsf{urPKE}})$
Else
  $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \xleftarrow{\$} \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
Require $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \bot$
If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
  $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
  $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
  $\mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r_{\mathsf{urPKE}})$
  $\mathrm{stR} \leftarrow (\mathrm{dk}, \mathrm{vk})$
  If $(c, \mathrm{ad}) \neq \boldsymbol{cad}[r]$:
    $\mathrm{is} \leftarrow \mathtt{fal}$
  If $(c, \mathrm{ad}) = \boldsymbol{cad}[r]$:
    $\boldsymbol{rcvd} \xleftarrow{\cup} \{r\}$
    $r \leftarrow r + 1$
If $\mathrm{is} = \mathtt{tru}$:
  $(\_, \mathrm{stR}) \leftarrow \boldsymbol{I}[r]$
Return

Oracle $\mathtt{Expose}_R$

If $\mathrm{is} = \mathtt{tru}$:
  Require $\boldsymbol{cc} \subseteq \boldsymbol{rcvd}$
  $\mathrm{xR} \leftarrow \mathtt{tru}$
Return stR

Fig. 25. Adversary $\mathcal{B}_q$ against IND-C security of urPKE for the proof of Lemma 13.

states of both utopian games fulfill correctness. Thus the adversary must not call $\mathtt{Expose}_S$ or $\mathtt{ChallExpose}_S$ for the same instance prior to the call to $\mathtt{ChallExpose}_R$. Further, the adversary must not call $\mathtt{Expose}_S$ or $\mathtt{ChallExpose}_S$ after the call to $\mathtt{ChallExpose}_R$(violates matching). Since the adversary must not call any sender state exposure oracles and since there are no non-trivial, by robustness the adversary can not call oracle Rcv s.t. the game's receiver state changes. Since the adversary must not call any sender state exposure oracles, oracle RR does not yield any useful information and is independent of the game's receiver state. Since the adversary can only call oracle RR after the only call to oracle $\mathtt{ChallExpose}_R$ it

```
Game G_0^b = ANON^b, G_1^b, G_2^b, G_3^0, G_4^0, G_5^0, G_6^0, G_7^0, G_8^0
I ← [·]
For i ∈ [q_S + q_CS]:
   I.append(RKE.init())
(stS, stR) ← I[0]
ceStR ← ⊥
b' ←$ A
Stop with b'

Oracle Snd(ad)
(stS, _) ← I[s_b]                                   //G2
(stS_next, (_, vk_next)) ← I[s_b + 1]               //G2
(stS_next, (_, vk_next)) ← I[s_1 + 1]               //G8
(ek, sk) ← stS
k_H, k_S ←$ K
(vk_next, sk_next) ← urSIG.gen
c_urPKE ←$ urPKE.enc(ek, (k_H, k_S, vk_next))
σ ←$ urSIG.sig(sk, (c_urPKE, ad))
c ← (c_urPKE, σ ⊕ PRG(k_S))
(k, r_urPKE, r_urSIG) ← H(k_H, c, ad)
sk ← urSIG.nextSk(sk_next, r_urSIG)
ek ← urPKE.nextEk(ek, r_urPKE)
stS ← (ek, sk)
stS ← stS_next                                      //G2
Return (c, k)

Oracle Rcv(c, ad)
k ← ⊥
(dk, vk) ← stR
(_, (dk, vk)) ← I[r_b]                              //G2
(_, (dk, vk)) ← I[r_1]                              //G8
(c_urPKE, σ') ← c
(k_H, k_S, vk_next) ← urPKE.dec(dk, c_urPKE)
Require (k_H, k_S, vk_next) ≠ ⊥
If urSIG.vfy(vk, (c_urPKE, ad), σ' ⊕ PRG(k_S))
   (k, r_urPKE, r_urSIG) ← H(k_H, c, ad)            //G0
   (k, r_urPKE, r_urSIG) ← h[k_H, c, ad]            //G1
   vk ← urSIG.nextVk(vk_next, r_urSIG)
   dk ← urPKE.nextDk(dk, r_urPKE)
   stR ← (dk, vk)
   (_, stR) ← I[r_b + 1]                            //G2
   (_, stR) ← I[r_1 + 1]                            //G8
Return [[k ≠ ⊥]]:

Oracle ChallSnd(ad)
(stS, _) ← I[s_b]                                   //G2
(stS, _) ← I[s_1]                                   //G8
(stS_next, (_, vk_next)) ← I[s_b + 1]               //G2
(stS_next, (_, vk_next)) ← I[s_1 + 1]               //G8
If b = 1:
   (stS, _) ←$ RKE.init
(ek, sk) ← stS
k_H, k_S ←$ K
(vk_next, sk_next) ← urSIG.gen
c_urPKE ←$ urPKE.enc(ek, (k_H, k_S, vk_next))
c_urPKE ←$ G^2                                      //G4
σ ←$ urSIG.sig(sk, (c_urPKE, ad))
c ← (c_urPKE, σ ⊕ PRG(k_S))
c ← (c_urPKE, ←$ S)                                 //G5
(k, r_urPKE, r_urSIG) ← H(k_H, c, ad)
sk ← urSIG.nextSk(sk_next, r_urSIG)
ek ← urPKE.nextEk(ek, r_urPKE)
If b = 0:
   stS ← (ek, sk)
   stS ← stS_next                                   //G2

Return (c, k)

Oracle H(x)
If h[x] ≠ ⊥:
   ABORT                                            //G1
   Return h[x]
h[x] ←$ G × K × K
Return h[x]

Oracle RR
(ek, sk) ← stS
ek ←$ urPKE.rr(ek)
sk ←$ urSIG.rr(sk)
stS ← (ek, sk)
Return

Oracle Expose_S
Return stS

Oracle ChallExpose_S
(ek, sk) ← stS
((ek', sk'), ceStR) ←$ RKE.init()
Return (ek', sk')                                   //G7
Return (ek', sk)                                    //G6
If b = 0:
   Return stS
Return stS'

Oracle Expose_R
Return stR

Oracle ChallExpose_R
(_, stR') ←$ RKE.init()
Return stR'                                         //G3
If b = 0:
   Return stR
Return stR'
```

Fig. 26. Games for the proof of Theorem 1.

can not learn any differences in the receiver state. Thus the output of oracle $\texttt{ChallExpose}_R$ is statistically indistinguishable from random.

Assume the adversary tries to impersonate the random world and then call oracle $\texttt{ChallExpose}_R$. Recall that for the adversary to impersonate the random world it must expose the sender state of the random world utopian game and compute the RKE.snd function on it. By definition of the requirements for matching trivial attacks, the adversary must not call any expose sender oracle after a call to $\texttt{ChallExpose}_R$. So the sender corruption must take place beforehand. Recall that prior to calling oracle $\texttt{ChallExpose}_R$, the adversary must not know the real world sender state matching the real world receiver state. So prior to exposing the receiver, the adversary must advance the real world utopian game's receiver state to the point such that an call to $\texttt{ChallExpose}_R$ can not be used any more to match that previously exposed sender state in the real world. In order to impersonate the random world, the adversary must not advance the random world utopian games's receiver state and must corrupt the sender state with an expose oracle which also returns the random world utopian game's receiver state. Thus the only strategy left for the adversary is to corrupt the sender via a call to oracle $\texttt{Expose}_S$, call oracle $\texttt{ChallSnd}(\mathrm{ad}) \to c$ on any additional data ad and deliver all ciphertexts including (c, ad) in order to oracle Rcv before calling oracle $\texttt{ChallExpose}_R$. Any future calls to oracles $\texttt{Expose}_S$ and $\texttt{ChallExpose}_S$ are prohibited, as their outputs match the receiver state the adversary learned as output of $\texttt{ChallExpose}_R$ only in the real world. Since the adversary must not call any sender state exposure oracles, oracle RR does not yield any useful information and is independent of the game's receiver state. Thus the outputs the adversary learns from oracle $\texttt{ChallExpose}_R$ are independent from any other value the adversary sees.

Since the output of oracle $\texttt{ChallExpose}_R$ is statistically independent of every other output, $\Pr\left[\mathsf{G}_2^{\mathcal{A}} \Rightarrow 1\right] = \Pr\left[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1\right]$.

Experiment Exp_4. In this game we replace values c_{urPKE} in oracle $\texttt{ChallSnd}$ with values drawn uniformly at random from the ciphertext space. By a standard hybrid argument over the number of queries to oracles $\texttt{Snd}$ and $\texttt{ChallSnd}$, we show in Lemma 14 that the advantage of an adversary distinguishing the real versions of this game from the previous games is upper bounded as, $\left|\Pr\left[\mathsf{G}_3^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_4^{\mathcal{A}} \Rightarrow 1\right]\right| \leq (q_S + q_{CS}) \cdot \mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{urPKE}}$.

Note that the random world versions are already statistically equivalent.

To show that no adversary can distinguish this game and the prior game we argue as follows. We first exclude the case where the adversary does not call oracle $\texttt{Expose}_R$. If so, the adversary can not observe any state changes due to impersonations. So for this case we give an reduction of the indistinguishability of both games to ANON of urPKE in Lemma 14.

Assume the adversary calls oracle $\texttt{Expose}_R$ on some instance. Since the utopian game is run on independent RKE instances, the exposure of the receiver returns a receiver state which is statistically independent from prior receiver states.

We now distinguish all combinations of impersonations. If there were no impersonations at all then by the definitions of our requirements against decryptability trivial attacks, all challenge ciphertexts must be received successfully before calling $\texttt{Expose}_R$. In our construction this means that the current receiver state instance must be statistically independent of prior instances. Thus if the adversary calls $\texttt{Expose}_R$ on some instance then the adversary must not call oracle $\texttt{ChallSnd}$ on this instance and any subsequent instances. Further, by definition of the trivial attacks there must be no prior exposed sender states which only match in one world the current receiver state. Since the current receiver state is instance separated this translates in our construction to the requirement that the adversary must not have called $\texttt{ChallExpose}_S$ on this and any subsequent instance. Since the adversary neither called $\texttt{ChallSnd}$ nor $\texttt{ChallExpose}_S$ on this or any subseqnet instances, the adversary can not distinguish.

If the adversary impersonated exclusively one world then the adversary must have known a sender state which matches the current receiver state in exclusively one world. By definition of our requirements against trivial matching attacks the adversary never learns such a sender state. Thus this case does not occur.

If the adversary impersonated both worlds then the receiver states in both worlds can neither be used to decrypt ciphertexts produced in any world nor to match sender states in any world. Thus the outputs of oracles $\texttt{ChallSnd}$ and $\texttt{ChallExpose}_S$ are statistically indistingusihable from uniform randomness.

Lemma 14. *Let $\mathcal{A}$ be an adversary which distinguishes games G_3 and G_4. We show how to construct an adversary $\mathcal{B}$, s.t. the advantage of adversary $\mathcal{A}$ is upper bounded by $(q_S + q_{CS}) \cdot \mathsf{Adv}^{\mathsf{ANON}}_{\mathsf{urPKE}}$.*

Proof. Let $\mathcal{A}$ be an adversary distinguishing G_3 and G_4. Assume that $\mathcal{A}$ does not call oracle $\texttt{Expose}_R$. We then show how to construct an adversay $\mathcal{B}$ against ANON of urPKE.

To this end we give a hybrid argument where hybrids are defined over every independent RKE instance. So in Fig. 27 we introduce hybrids H_q, where $q \in [(q_S + q_{CS})]$. We define the hybrids s.t. $\mathsf{H}_0 := \mathsf{G}_4$ and $\mathsf{H}_{(q_S+q_{CS})} := \mathsf{G}_3$.

We show in Fig. 28 how to construct adversaries $\mathcal{B}_q$ which bound the advantage of any adversary distinguishing hybrids H_q and H_{q+1}, with the advantage of breaking $\mathsf{ANON}_{\mathsf{urPKE}}$.

In the following we show that if adversary $\mathcal{B}_q$ is executed in the real or random version of $\mathsf{ANON}_{\mathsf{urPKE}}$ then $\mathcal{B}_q$ simulates the hybrid H_q or H_{q+1}, respectively.

Since every call to oracle $\texttt{Snd}$ and $\texttt{ChallSnd}$ makes the sender state statistically independent from previous sender states, adversary $\mathcal{B}$ can embed ek^* in the sender state after the $q-1$th query to oracles $\texttt{Snd}$ and $\texttt{ChallSnd}$. Adversary $\mathcal{B}$ checks whether the current call to oracle $\texttt{Snd}$ or $\texttt{ChallSnd}$ was the qth call by comparing q and s_0.

To answer queries to oracle $\texttt{Snd}$, adversary $\mathcal{B}$ can use ek^* in the qth query to produce ciphertext c_{urPKE}.

To answer queries to oracle $\texttt{ChallSnd}$, adversary $\mathcal{B}$ uses oracle $\mathsf{ChallSnd}$ provided by the $\mathsf{ANON}_{\mathsf{urPKE}}$ security experiment.

To answer queries to oracle $\texttt{Rcv}$ the adversary $\mathcal{B}$ does the following. We distinguish the inputs to oracle $\texttt{Rcv}$. If the inputs to oracle $\texttt{Rcv}$ are in order and there are no impersonations until the qth ciphertext output by oracles $\texttt{Snd}$ and $\texttt{ChallSnd}$ then adversary $\mathcal{B}$ must by able to decrypt a c_{urPKE} ciphertext for the decryption key matching ek^*. To do so it calls oracle Dec provided by the $\mathsf{ANON}_{\mathsf{urPKE}}$ security experiment. To check for the qth ciphertext produced by the game, $\mathcal{B}$ checks whether $r_0 = q$. Recall that r_0 counts the number of successful receive operations in the real world execution. If prior to receiving the qth ciphertext there was an impersonation then adversary $\mathcal{B}$ produces the same distribution as the respective hybrid.

To simulate queries to oracles $\mathtt{Expose}_S$ and $\mathtt{ChallExpose}_S$, the adversary outputs on the qth instance ek^* and behaves indistinguishable from the hybrid distributions.

To simulate queries to oracle RR, adversary $\mathcal{B}$ behaves as follows. After the qth call to oracles Snd and ChallSnd, the adversary randomizes the encryption key by calling oracles RR and $\mathsf{Expose}_\mathsf{S}$.

Hybrids H_q

$\boldsymbol{I} \leftarrow [\cdot]$
For $i \in [q_S + q_{CS}]$:
  $\boldsymbol{I}$.append(RKE.init)
$(\mathrm{stS}, \mathrm{stR}) \leftarrow \boldsymbol{I}[0]$
$ceStR \leftarrow \bot$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle Snd(ad)

$(\mathrm{stS}, \_) \leftarrow \boldsymbol{I}[s_0]$
$(\mathrm{stS}_{\mathrm{next}}, (\_, \mathrm{vk}_{\mathrm{next}})) \leftarrow \boldsymbol{I}[s_0 + 1]$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_\mathsf{H}, k_S \xleftarrow{\$} \mathcal{K}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_\mathsf{H}, k_S, \mathrm{vk}_{\mathrm{next}}))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$\mathrm{stS} \leftarrow \mathrm{stS}_{\mathrm{next}}$
Return (c, k)

Oracle Rcv(c, ad)

$k \leftarrow \bot$
$(\_, (\mathrm{dk}, \mathrm{vk})) \leftarrow \boldsymbol{I}[r_0]$
$(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
$(k_\mathsf{H}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
Require $(k_\mathsf{H}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \bot$
If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
  $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow h[k_\mathsf{H}, c, \mathrm{ad}]$
  $(\_, \mathrm{stR}) \leftarrow \boldsymbol{I}[r_0 + 1]$
Return $[\![k \neq \bot]\!]$:

Oracle ChallSnd(ad)

$(\mathrm{stS}, \_) \leftarrow \boldsymbol{I}[s_0]$
$(\mathrm{stS}_{\mathrm{next}}, (\_, \mathrm{vk}_{\mathrm{next}})) \leftarrow \boldsymbol{I}[s_0 + 1]$
If $i > q$:
  $(\mathrm{stS}, \_) \xleftarrow{\$} \mathsf{RKE.init}$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_\mathsf{H}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}_{\mathrm{next}}, \mathrm{sk}_{\mathrm{next}}) \leftarrow \mathsf{urSIG.gen}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_\mathsf{H}, k_S, \mathrm{vk}_{\mathrm{next}}))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_\mathsf{H}, c, \mathrm{ad})$
$\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
$\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r_{\mathsf{urPKE}})$
If $i \leq q$:
  $\mathrm{stS} \leftarrow \mathrm{stS}_{\mathrm{next}}$
If $i > q$:
  $(\mathrm{stS},) \leftarrow \boldsymbol{I}[s_0]$
Return (c, k)

Oracle H(x)

If $h[x] \neq \bot$:
  ABORT
  Return $h[x]$
$h[x] \xleftarrow{\$} \mathbb{G} \times \mathcal{K} \times \mathcal{K}$
Return $h[x]$

Oracle RR

$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$
$\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$
$\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
Return

Oracle $\mathtt{Expose}_S$

Return stS

Oracle $\mathtt{ChallExpose}_S$

Return stS

Oracle $\mathtt{Expose}_R$

Return stR

Oracle $\mathtt{ChallExpose}_R$

$(\_, \mathrm{stR}') \xleftarrow{\$} \mathsf{RKE.init}$
Return stR'

Fig. 27. Hybrids H_q for the proof of Lemma 14.

Experiment Exp_5. In this game we replace the remaining output of oracle ChallSnd with values drawn uniformly at random from the respective space. By a standard hybrid argument one can upper bound $\left|\Pr\left[\mathsf{G}_4^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_5^{\mathcal{A}} \Rightarrow 1\right]\right|$ by q_{CS} times the advantage of breaking PRG security.

Experiment Exp_6. In this game we replace the secret key output by oracle $\mathtt{ChallExpose}_S$ with uniform randomness. To show that the adversary can not distinguish this game from the previous game we argue as follows.

By the same argument as given in G_4 argue that if the adversary would call oracle $\mathtt{Expose}_R$ it would see only statistically independent values. For the case where the adversary does not call oracle $\mathtt{Expose}_R$ we give an reduction of the indistinguishability of both games to IND-R of urSIG in Lemma 15.

Thus, $\left|\Pr\left[\mathsf{G}_5^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_6^{\mathcal{A}} \Rightarrow 1\right]\right| \leq q_{CE} \cdot \mathsf{Adv}_{\mathsf{urSIG}}^{\mathsf{IND\text{-}R}}$.

Lemma 15. *The advantage of any PPT adversary distinguishing between games* G_5 *and* G_6 *is bounded by* $q_{CE} \cdot \mathsf{Adv}_{\mathsf{urSIG}}^{\mathsf{IND\text{-}R}}$.

Proof. Let $\mathcal{A}$ be an adversary distinguishing G_5 and G_6. Assume that $\mathcal{A}$ does not call oracle $\mathtt{Expose}_R$. We then show how to construct an adversay $\mathcal{B}$ against IND-R of urSIG.

Adversary $\mathcal{B}_q^{\mathsf{ChallSnd},\mathsf{RR},\mathsf{Dec},\mathsf{Expose}_S}(\mathrm{ek}^*)$

$\boldsymbol{I} \leftarrow [\cdot]$
For $i \in [q_S + q_{CS}]$:
  $\boldsymbol{I}.\mathsf{append}(\mathsf{RKE.init})$
$(\mathrm{stS}, \mathrm{stR}) \leftarrow \boldsymbol{I}[s_0]$
$ceStR \leftarrow \bot$
$b' \xleftarrow{\$} \mathcal{A}$
Stop with b'

Oracle $\mathtt{Snd}(\mathrm{ad})$

$(\mathrm{stS}, \_) \leftarrow \boldsymbol{I}[s_0]$
$(\_, (\_, \mathrm{vk}_{\mathrm{next}})) \leftarrow \boldsymbol{I}[s_0 + 1]$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}))$
If $s_0 = q$:
  $c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}^*, (k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}))$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
If $s_0 = q - 1$:
  $((\mathrm{ek}', \mathrm{sk}'), \mathrm{stR}') \leftarrow \boldsymbol{I}[s_0 + 1]$
  $\mathrm{ek}' \leftarrow \mathrm{ek}^*$
  $\boldsymbol{I}[s_0 + 1] \leftarrow ((\mathrm{ek}', \mathrm{sk}'), \mathrm{stR}')$
$\mathrm{stS} \leftarrow (\mathrm{ek}', \mathrm{sk}')$
Return (c, k)

Oracle $\mathtt{Rcv}(c, \mathrm{ad})$

$k \leftarrow \bot$
$(\mathrm{dk}, \mathrm{vk}) \leftarrow \mathrm{stR}$
$(\_, (\mathrm{dk}'', \mathrm{vk}'')) \leftarrow \boldsymbol{I}[r_0 + 1]$
$(c_{\mathsf{urPKE}}, \sigma') \leftarrow c$
$(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathsf{urPKE.dec}(\mathrm{dk}, c_{\mathsf{urPKE}})$
If $r_0 = q \wedge \mathrm{imp}_0 = \mathtt{fal}$:
  $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \leftarrow \mathtt{Dec}(\mathrm{ctr}_{\mathsf{urPKE}}, c)$
Require $(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}) \neq \bot$
If $\mathsf{urSIG.vfy}(\mathrm{vk}, (c_{\mathsf{urPKE}}, \mathrm{ad}), \sigma' \oplus \mathsf{PRG}(k_S))$
  $(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow h[[k_{\mathsf{H}}, c, \mathrm{ad}]$
  $\mathrm{vk} \leftarrow \mathsf{urSIG.nextVk}(\mathrm{vk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
  $\mathrm{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathrm{dk}, r_{\mathsf{urPKE}})$
Return $[\![k \neq \bot]\!]$:

Oracle $\mathtt{ChallSnd}(\mathrm{ad})$

$(\mathrm{stS}, \_) \leftarrow \boldsymbol{I}[s_0]$
$(\mathrm{stS}_{\mathrm{next}}, (\_, \mathrm{vk}_{\mathrm{next}})) \leftarrow \boldsymbol{I}[s_0 + 1]$
If $s_0 > q$:
  $(\mathrm{stS}, \_) \xleftarrow{\$} \mathsf{RKE.init}$
$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$k_{\mathsf{H}}, k_S \xleftarrow{\$} \mathcal{K}$
$(\mathrm{vk}_{\mathrm{next}}, \mathrm{sk}_{\mathrm{next}}) \leftarrow \mathsf{urSIG.gen}$
$c_{\mathsf{urPKE}} \xleftarrow{\$} \mathsf{urPKE.enc}(\mathrm{ek}, (k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}}))$
If $s_0 = q$:
  $c_{\mathsf{urPKE}} \leftarrow \mathsf{ChallSnd}(k_{\mathsf{H}}, k_S, \mathrm{vk}_{\mathrm{next}})$
$\sigma \xleftarrow{\$} \mathsf{urSIG.sig}(\mathrm{sk}, (c_{\mathsf{urPKE}}, \mathrm{ad}))$
$c \leftarrow (c_{\mathsf{urPKE}}, \sigma \oplus \mathsf{PRG}(k_S))$
$(k, r_{\mathsf{urPKE}}, r_{\mathsf{urSIG}}) \leftarrow \mathsf{H}(k_{\mathsf{H}}, c, \mathrm{ad})$
$\mathrm{sk} \leftarrow \mathsf{urSIG.nextSk}(\mathrm{sk}_{\mathrm{next}}, r_{\mathsf{urSIG}})$
$\mathrm{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathrm{ek}, r_{\mathsf{urPKE}})$
If $s_0 \leq q$:
  $\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
  $\mathrm{stS} \leftarrow \mathrm{stS}_{\mathrm{next}}$
If $s_0 > q$:
  $(\mathrm{stS}, \_) \leftarrow \boldsymbol{I}[s_0]$
If $s_0 = q - 1$:
  $((\_, \mathrm{sk}), \_) \leftarrow \boldsymbol{I}[s_0 - 1]$
  $\mathrm{ek}' \leftarrow \mathrm{ek}^*$
  $\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
Return (c, k)

Oracle $\mathsf{H}(x)$

If $h[x] \neq \bot$:
  ABORT
  Return $h[x]$
$h[x] \xleftarrow{\$} \mathbb{G} \times \mathcal{K} \times \mathcal{K}$
Return $h[x]$

Oracle $\mathtt{RR}$

$(\mathrm{ek}, \mathrm{sk}) \leftarrow \mathrm{stS}$
$\mathrm{sk} \xleftarrow{\$} \mathsf{urSIG.rr}(\mathrm{sk})$
If $s_0 \neq q$:
  $\mathrm{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathrm{ek})$
  $\mathrm{stS} \leftarrow (\mathrm{ek}, \mathrm{sk})$
If $s_0 = q$:
  $\mathsf{RR}()$
  $\mathrm{ek}^* \leftarrow \mathsf{Expose}_S$
  $\mathrm{stS} \leftarrow (\mathrm{ek}^*, \mathrm{sk})$
Return

Oracle $\mathtt{Expose}_S$

If $s_0 \neq q$:
  Return stS
If $s_0 = q$:
  Return $(\mathrm{ek}^*, \mathrm{sk})$

Oracle $\mathtt{ChallExpose}_S$

If $s_0 \neq q$:
  Return stS
If $s_0 = q$:
  Return $(\mathrm{ek}^*, \mathrm{sk})$

Oracle $\mathtt{Expose}_R$

Return stR

Oracle $\mathtt{ChallExpose}_R$

$(\_, \mathrm{stR}') \xleftarrow{\$} \mathsf{RKE.init}$
Return stR'

Fig. 28. Adversary $\mathcal{B}_q$ against ANON security of urPKE for the proof of Lemma 14.

In Fig. 29 we introduce hybrids H_q, where $q \in [q_{CE}]$. We define the hybrids s.t. $\mathsf{H}_0 := \mathsf{G}_6$ and $\mathsf{H}_{q_{CE}} := \mathsf{G}_5$.

Clearly the advantage differentiating hybrids H_q and H_{1+q} is bounded by $\mathsf{IND\text{-}R}_{\mathsf{urSIG}}$.

Experiment Exp_7. In this game we replace the encryption key output of oracle $\mathtt{ChallExpose}_S$ with uniform randomness. Similar to the last game we exclude the case, where the adversary called oracle $\mathtt{Expose}_R$ and give a reduction of the indistinguishability of both games to $\mathsf{IND\text{-}R}$ of urPKE in Lemma 16.

Thus, $\left|\Pr\left[\mathsf{G}_6^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_7^{\mathcal{A}} \Rightarrow 1\right]\right| \leq q_{CE} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{A},\mathsf{urPKE}_{\mathsf{EG}}}$.

Lemma 16. *The advantage of any PPT adversary distinguishing between games* G_6 *and* G_7 *is bounded by* $q_{CE} \cdot \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{A},\mathsf{urPKE}_{\mathsf{EG}}}$.

Proof. Let $\mathcal{A}$ be an adversary distinguishing G_6 and G_7. Assume that $\mathcal{A}$ does not call oracle $\mathtt{Expose}_R$. We then show how to construct an adversay $\mathcal{B}$ against $\mathsf{IND\text{-}R}$ of urSIG.

In Fig. 29 we introduce hybrids I_q, where $q \in [q_{CE}]$. We define the hybrids s.t. $\mathsf{I}_0 := \mathsf{G}_7$ and $\mathsf{I}_{q_{CE}} := \mathsf{G}_6$.

Clearly the advantage differentiating hybrids I_q and I_{1+q} is bounded by $\mathsf{IND\text{-}R}_{\mathsf{urSIG}}$.

```
Hybrid H_q, [I_q]
I ← [·]
For i ∈ [q_S + q_CS]:
    I.append(RKE.init())
j ← 0
(stS, stR) ← I[0]
ceStR ← ⊥
b' ←$ A
Stop with b'

Oracle Snd(ad)
(stS, _) ← I[s_0]
(_, (_, vk_next)) ← I[s_0 + 1]
(ek, sk) ← stS
k_H, k_S ←$ K
(vk_next, sk_next) ← urSIG.gen
(stS', (_, vk_next)) ← I[s_b]
c_urPKE ←$ urPKE.enc(ek, (k_H, k_S, vk_next))
σ ←$ urSIG.sig(sk, (c_urPKE, ad))
c ← (c_urPKE, σ ⊕ PRG(k_S))
(k, r_urPKE, r_urSIG) ← H(k_H, c, ad)
sk ← urSIG.nextSk(sk_next, r_urSIG)
ek ← urPKE.nextEk(ek, r_urPKE)
stS ← (ek, sk)
stS ← stS'
Return (c, k)

Oracle ChallSnd(ad)
(stS', _) ←$ RKE.init
(_, c, k) ←$ RKE.snd(stS', ad)
Return (c, k)

Oracle Rcv(c, ad)
k ← ⊥
(dk, vk) ← stR
(_, (dk'', vk'')) ← I[r_b]
(c_urPKE, σ') ← c
(k_H, k_S, vk_next) ← urPKE.dec(dk, c_urPKE)
Require (k_H, k_S, vk_next) ≠ ⊥
If urSIG.vfy(vk, (c_urPKE, ad), σ' ⊕ PRG(k_S))
    (k, r_urPKE, r_urSIG) ← h[k_H, c, ad]
    vk ← urSIG.nextVk(vk_next, r_urSIG)
    dk ← urPKE.nextDk(dk, r_urPKE)
    stR ← (dk, vk)
Return [[k ≠ ⊥]]:

Oracle H(x)
If h[x] ≠ ⊥:
    ABORT
h[x] ←$ G × K × K
Return h[x]

Oracle RR
(ek, sk) ← stS
ek ←$ urPKE.rr(ek)
sk ←$ urSIG.rr(sk)
stS ← (ek, sk)
Return

Oracle Expose_S
Return stS

Oracle ChallExpose_S
(ek, sk) ← stS
((ek', sk'), ceStR) ←$ RKE.init()
If j ≤ q:
    Return (ek, sk)
If j > q:
    [Return (ek', sk')]
    Return (ek', sk')
If b = 0:
    Return stS
Return stS'

Oracle Expose_R
Return stR

Oracle ChallExpose_R
(_, stR') ←$ RKE.init()
Return stR'
```

Fig. 29. Hybrids H_q, I_q for the proof of Lemmas 15 and 16.

Experiment Exp_8. In this game we set s_b to s_1 and r_b to r_1. Effectively this sets number of sender state updates by oracle `ChallSnd` and the receiver state updates by receiving in-order challenges in oracle `Rcv` in both worlds alike.

We now argue that there exists no adversary which distinguishes this game from G_7. For the adversary to observe the change in both games it must query oracle `ChallSnd`. By definition of our trivial attacks the adversary must not call oracle $\texttt{Expose}_R$ on that instance. Recall that the outputs of oracle $\texttt{ChallExpose}_S$ are indistinguishable from random to the adversary. Since the output of oracle `ChallSnd` itself is indistinguishable from random to the adversary, the adversary can only observe a difference in the updates of the sender state if it queries on a single instance i)$\texttt{Expose}_S$ ii)a nearly arbitrary combinations of oracles `RR` and `ChallSnd` iii)$\texttt{Expose}_S$. The only restriction for ii) is that the adversary must query `RR` after the last query to oracle `ChallSnd`. If so oracle `RR` randomizes the sender state in both worlds. Thus by IND-R of urPKE and by IND-R of urSIG the distribution of both exposed sender states is indistinguishable from random. By similar hybrids as in the last two games, one can show that, $\left|\Pr\left[\mathsf{G}_7^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathsf{G}_8^{\mathcal{A}} \Rightarrow 1\right]\right| \leq q_{CS} \cdot \left(\mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathcal{A},\mathsf{urPKE}_{\mathsf{EG}}} + \mathsf{Adv}^{\mathsf{IND\text{-}R}}_{\mathsf{urSIG}}\right)$.

G Attacks Against RKE Constructions

Existing constructions of RKE fail to fulfill the requirements of our anonymity notion for several reasons. Most trivially, multiple RKE constructions in the literature are already insecure with respect to key secrecy (see Definition 11), which also leads to attacks against anonymity. Beyond that, many constructions that offer *strong* key secrecy and authenticity have highly structured ciphertexts or let senders sign these ciphertexts. Without using advanced tools comparable to urSIG, this breaks anonymity, too.

As mentioned in the introduction, we refrain from breaking anonymity of RKE schemes designed for the group setting. It is meaningless to present (non-trivial) attacks against anonymity of CGKA (or "group RKE") constructions without having a satisfiable definition for the group setting that separates non-trivial attacks from trivial ones. Nevertheless, it is obvious that all known CGKA constructions leak information via (publicly) sent ciphertexts and exposed user states, which allows for identifying the respective CGKA session or even a specific user in a session. This is particularly true for all CGKA constructions that rely on an actively participating server.

Constructions with Weak Secrecy. The most prominent RKE protocol, the Double Ratchet Algorithm [PM16], only relies on a symmetric hash-chain for unidirectional communication. That is, as long as Alice only sends to Bob without receiving a reply, she will continuously compute each new symmetric session key deterministically from the respective prior one. Thus, by exposing Alice's state before she sends, an adversary can pre-compute her next state, which violates our notion of anonymity. However, we want to note that Vatandas et al. [VGIK20] prove deniability for the Double Ratchet Algorithm, which means that Alice can deny her participation in a session if her state is exposed *after* the session terminated.

The unidirectional RKE protocol proposed by Bellare et al. [BSJ+17] overcomes the limitation of the Double Ratchet Algorithm by sampling asymmetric keys for every send operation and probabilistically updating the sender state. However, as discovered by Poettering and Rösler [PR18b, PR18a], the construction in [BSJ+17] does not offer forward-secrecy on the receiver side. For this, consider an adversary who first exposes Alice's sender state, then lets Alice send a couple of ciphertext $(c_i)_{i\in[l]}$ one after another and forwards all of them honestly to Bob, who receives them. Finally, the adversary exposes Bob's receiver state. This adversary can compute all symmetric session keys, established by the transmitted ciphertexts. The reason for this is that Bob has an *almost* static receiver state. Coming back to anonymity, this means that the same adversary can also verify whether the ciphertexts seen on the network were sent from Alice to Bob or whether they were sent in an independent RKE session, which again violates anonymity.

Constructions with Structured States and Ciphertexts. Constructions with stronger secrecy and anonymity guarantees, such as [PR18b, JS18, JMM19a, JMM19b, BRV20, CDV21], fail to achieve anonymity because it is trivial to link their ciphertexts within single sessions, or link ciphertexts with the corresponding sender states.

TRACING CIPHERTEXTS. Many constructions [JS18, JMM19a, JMM19b, CDV21] embed a continuously incremented integer (counting the number of send operations) or a transcript hash in the ciphertexts sent from Alice to Bob. Both values, especially a publicly computable transcript hash, can be used to identify a session and its participants, even without exposing their local states.

TRACING STATES. While not all constructions *publicly* reveal transcript hashes or counters in the sent ciphertexts, all constructions [PR18b, JS18, JMM19a, JMM19b, BRV20, CDV21] let senders store this information locally in the *secret* sender states. Since counters or publicly computable transcript hashes can be precomputed (see our attack against anonymity of the Double Ratchet Algorithm), verifying that two exposed states belong to the same sender breaks anonymity.

TRACING VIA AUTHENTICATION. Finally, as mentioned in the introduction (see Section 1.1), employed authentication mechanisms can reveal the sender of a ciphertext. For example, using the symmetric message authentication key exposed from the sender state [PR18b, BRV20], an adversary can simply verify the message authentication tag of a subsequently sent ciphertext, which breaks anonymity. Similarly, the signing key exposed from the sender state [JS18, JMM19a, JMM19b, CDV21] can (usually) be used to verify a signature of a subsequently sent ciphertext.

Our construction overcomes all these shortcomings by having a re-randomizable sender state and a randomly looking receiver state. Furthermore, ciphertexts on the network do not reveal a relation to their senders, even if these senders were exposed before or afterwards.

Although, we may have missed a published RKE construction when compiling the list of related articles from the literature, we believe that at least one of the presented attack strategies is successful against all known constructions.