Strongly Anonymous Ratcheted Key Exchange

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

[1] University of Sheffield, Sheffield, UK
b.dowling@sheffield.ac.uk
[2] Ruhr-Universität Bochum, Bochum, Germany
{eduard.hauck,doreen.riepel}@rub.de
[3] New York University, New York, USA
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

---

The full version of this article is available in the IACR eprint archive as article 2022/1187, at https://eprint.iacr.org/2022/1187.

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 [32], *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 [5,8,34], 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., [25,33,34]), RKE with *immediate decryption* (e.g., [3]), RKE in *static* groups (e.g., [14]) and *dynamic* groups (e.g., [4,9,37]), resilient to *concurrent* operations (e.g., [2,10]), etc. In the full version of this paper [17], 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., [7,23,27]) to anonymous signatures (e.g., [21,41]) to privacy-preserving key exchange (e.g., [24,38,42])

[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.

to anonymous onion encryption (e.g., [15, 35]) 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., [15]), 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.

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 [39] 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 [30, 40] undermine its anonymity guarantees. The Sealed Sender mechanism is related to instances of the Noise protocol framework [18, 31] 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 [6] that is yet to be concluded.[6] Related to this, Emura et al. [20] informally propose changes to an early version of MLS by Cohn-Gordon et al. [14] 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 [4, 16, 26]. 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.

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

Due to the page limit, 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 full version [17].

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) [5,8,34]. 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 $\mathsf{RKE.init} \rightarrow_{\$} (\mathrm{stS}, \mathrm{stR})$. (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 $\mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad}) \rightarrow_{\$} (\mathrm{stS}, c, k)$ and the other user (Bob) receives these ciphertexts to compute these (same) established keys $\mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad}) \rightarrow_{\$} (\mathrm{stR}, k)$. Either way, secrets in both users' states are continuously renewed by these operations.

Standard Security Goals. Secrecy and authenticity of established symmetric keys for URKE have been studied in prior work [5,8,34]. 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 [34], 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 [19] requires that Bob must not falsely accept 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 [19] 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. [5] 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) [19, 25, 34]. 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 the full version [17].) 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: `Snd`, `RR`, `Rcv`, $\mathtt{Expose}_S$, $\mathtt{Expose}_R$. Internally, these oracles execute Alice's RKE.snd algorithm, outputting the resulting ciphertext, Alice's RKE.rr algorithm, Bob's 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 `RR` for the additional 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:

- `ChallSnd` equals oracle `Snd` iff $b = 0$; otherwise, it temporarily initializes a new, independent URKE session with algorithm RKE.init, uses the temporary sender to send a ciphertext with algorithm RKE.snd, and outputs this ciphertext (the temporary URKE session is discarded immediately afterwards); oracle `Rcv` silently ignores ciphertexts created by `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.

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

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.

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.

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

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 $\mathsf{ek} \leftarrow (g^r, g^{xr})$, where r and x are random exponents and $x = \mathsf{dk}$ is the decryption key. For re-randomizing the encryption key, we apply the same random exponent r' to both of its components $(\mathsf{ek}_0^{r'}, \mathsf{ek}_1^{r'})$. Encryption of message m takes a random exponent s to create ciphertext $c \leftarrow (\mathsf{ek}_0^s, \mathsf{H}(\mathsf{ek}_0^s, \mathsf{ek}_1^s) \oplus m)$. Decryption follows immediately via $m \leftarrow \mathsf{H}(c_0, c_0^{\mathsf{dk}}) \oplus c_1$.

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 [4,16,26] 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 [28]. Intuitively, we start generating the signing key by sampling $2 \cdot \ell$ pre-images $\mathsf{sk}'_{i,b}, (i, b) \in [\ell] \times \{0, 1\}$. To derive the matching verification key, we apply a one-way function on each pre-image $\mathsf{vk}'_{i,b} \leftarrow \mathsf{f}(\mathsf{sk}'_{i,b})$. Finally, we generate a PKE key pair $(\mathsf{ek}, \mathsf{dk})$ that allows ciphertext re-randomization. The final verification key consists of the decryption key dk and all images $\mathsf{vk}'_{i,b}$. The final signing key consists of the encrypted pre-images $\mathsf{sk}_{i,b} \leftarrow \mathsf{rPKE.enc}(\mathsf{ek}, \mathsf{sk}'_{i,b})$. To re-randomize Alice's verification key, she re-randomizes each component ciphertext $\mathsf{sk}_{i,b}$. The signature of message $m = (m_1, \ldots, m_\ell)$ consists of the respective signing key components $\sigma \leftarrow (\mathsf{sk}_{1,m_1}, \ldots, \mathsf{sk}_{\ell,m_\ell})$. 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 [13, 29] 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 $(\mathsf{sk}_{1,m_1}, \ldots, \mathsf{sk}_{\ell,m_\ell})$ to obtain a compact signature σ; this aggregation is inspired by BLS signatures [11, 12]. The full details of this construction are in Section 6.

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 the full version [17].

3 Ratcheted Key Exchange

Throughout this paper, we consider unidirectional communication, as defined in several flavors in previous works [5, 8, 34]. 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 exchange.

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 $\perp$.
- $\mathrm{stS} \xleftarrow{\$} \mathsf{RKE.rr}(\mathrm{stS})$ 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 $\mathsf{RKE.snd}$ algorithm. Let $\mathcal{AD} := \{0,1\}^*$ be the space of associated data.

State Randomization. All algorithms except $\mathsf{RKE.rr}$ 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 $\mathsf{RKE.snd}$ with one of her

states, she can re-randomize all remaining states via $\mathsf{RKE.rr}$. 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 the full version [17], we formally 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 $k = \bot$), 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 the full version [17]. 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 $\texttt{Expose}_S$ and $\texttt{Expose}_R$, respectively.

Secrecy. In game KIND, which models secrecy of session keys, the adversary can additionally query `ChallSnd`. This oracle internally executes algorithm $\mathsf{RKE.snd}$ 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 $\texttt{Expose}_R$ 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*.

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.

[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.

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 [36], 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. 1. 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 $\mathsf{U\text{-}ANON}^b_{\mathsf{RKE}}(\mathcal{A})$
00 $(\mathrm{stS}, \mathrm{stR}) \xleftarrow{\$} \mathsf{RKE.init}$
01 $\mathrm{ceStR} \leftarrow \bot$
02 $b' \xleftarrow{\$} \mathcal{A}$
03 Stop with b'

Oracle `Snd`(ad)
04 $(\mathrm{stS}, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad})$
05 Return (c, k)

Oracle `Rcv`(c, ad)
06 $(\mathrm{stR}, k) \leftarrow \mathsf{RKE.rcv}(\mathrm{stR}, c, \mathrm{ad})$
07 Return $[\![k \neq \bot]\!]$:

Oracle `RR`
08 $\mathrm{stS} \xleftarrow{\$} \mathsf{RKE.rr}(\mathrm{stS})$
09 Return

Oracle `ChallSnd`(ad)
10 If $b = 0$:
11 $\quad(\mathrm{stS}, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}, \mathrm{ad})$
12 If $b = 1$:
13 $\quad(\mathrm{stS}', \_) \xleftarrow{\$} \mathsf{RKE.init}$
14 $\quad(\_, c, k) \xleftarrow{\$} \mathsf{RKE.snd}(\mathrm{stS}', \mathrm{ad})$
15 Return (c, k)

Oracle $\texttt{Expose}_S$
16 Return stS

Oracle $\texttt{ChallExpose}_S$
17 If $b = 0$:
18 $\quad$Return stS
19 $(\mathrm{stS}', \mathrm{ceStR}) \xleftarrow{\$} \mathsf{RKE.init}$
20 Return stS'

Oracle $\texttt{Expose}_R$
21 Return stR

Oracle $\texttt{ChallExpose}_R$
22 If $b = 0$:
23 $\quad$Return stR
24 $(\_, \mathrm{stR}') \xleftarrow{\$} \mathsf{RKE.init}$
25 Return stR'

Fig. 1. 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. 2). 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. the full version [17]).

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. 1). 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 $\mathsf{U\text{-}ANON}_b.\texttt{Snd}(\text{ad})$, 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 $\mathsf{U\text{-}ANON}_b.\text{stR}$ 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 $\mathsf{U\text{-}ANON}_b.\text{stR}$ 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 $\boldsymbol{cc}$.

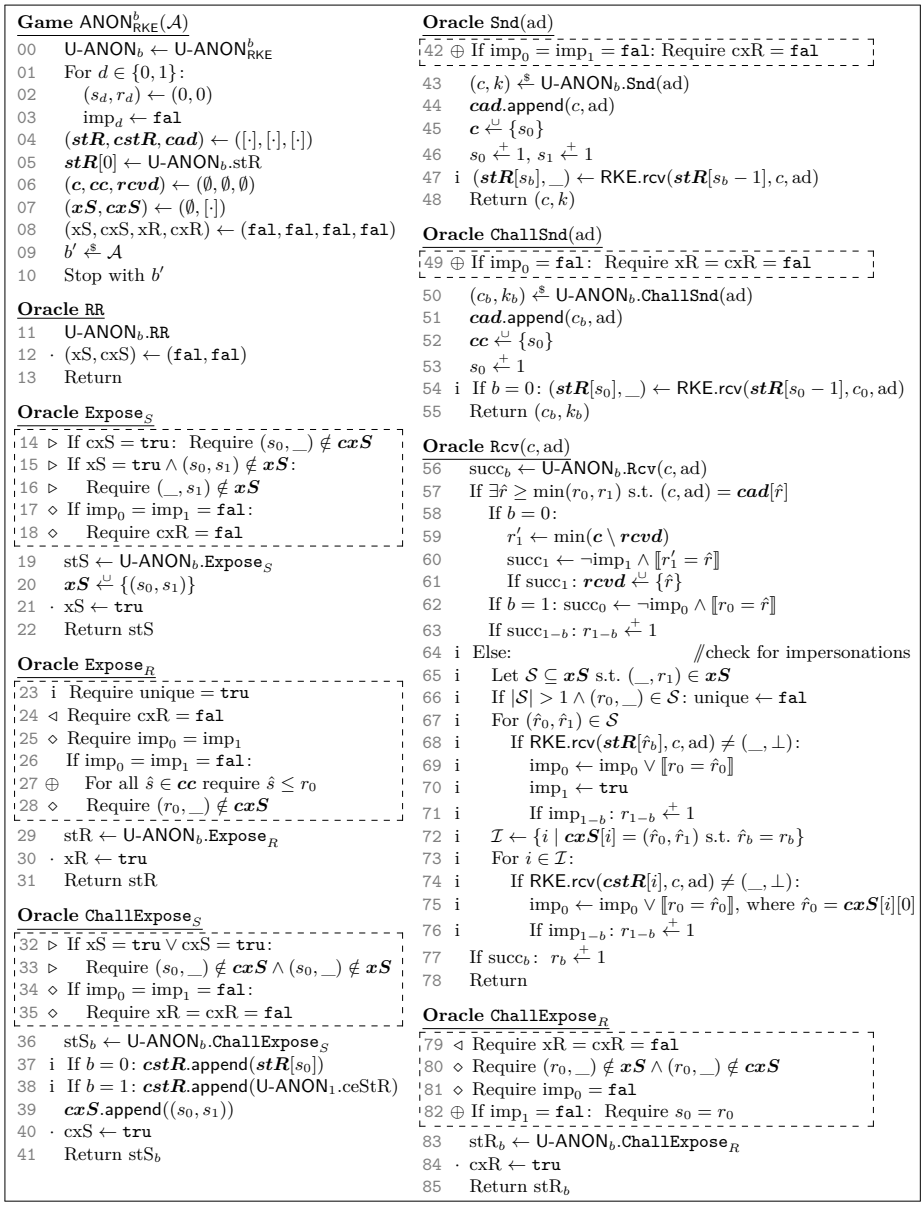

Fig. 2. Full anonymity games ANON^b for $b \in \{0,1\}$, where lines in dashed boxes disallow trivial attacks. We further distinguish between different trivial attacks (cf. Section 4.2): Lines marked with $\oplus$ are due to correctness relations, those marked with $\triangleright$, $\triangleleft$ are due to state equality relations on sender resp. receiver side, those marked with $\diamond$ are due to matching state relations, and i indicates an impersonation requirement.

Exposures and Randomizations. Oracles $\texttt{Expose}_S$ and $\texttt{Expose}_R$ 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 $\texttt{ChallExpose}_S$. 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 $\mathsf{U\text{-}ANON}_1$.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.

Receive Queries. Oracle `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 $\mathsf{U\text{-}ANON}_1$. 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 $\mathsf{U\text{-}ANON}_{1-b}$. 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

[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`, $\texttt{Expose}_S$ 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.

this query would lead to a successful receive in $\mathsf{U\text{-}ANON}_1$. At this point, we only care about ciphertexts from `Snd` since challenge ciphertexts in $\mathsf{U\text{-}ANON}_1$ 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 $\mathsf{U\text{-}ANON}_1$ 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 $\mathsf{U\text{-}ANON}_0$, 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 $\mathsf{U\text{-}ANON}_1$ 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 $\mathsf{U\text{-}ANON}_1$, so we set imp_1 to `tru`. If $\hat{r}_0 = r_0$, then we had an impersonation in $\mathsf{U\text{-}ANON}_0$ 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 the full version [17]. 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 $\mathsf{U\text{-}ANON}_0$ has occurred if the counter $\hat{r}_0$ in $\boldsymbol{cxS}$ equals the current counter r_0. Note that in $\mathsf{U\text{-}ANON}_1$, 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

[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.

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 the full version [17].

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. 2. We define the advantage of an adversary* $\mathcal{A}$ *against anonymity of a ratcheted key exchange scheme* RKE *as*

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

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 $(\mathrm{ek}, \_) \in \mathsf{rPKE.gen}$, 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,

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

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*

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

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

Construction. In Fig. 3, we construct an rPKE scheme based on the ElGamal KEM and PKE scheme. Thus, we denote the corresponding scheme by $\mathsf{rPKE}_{\mathsf{EG}}$. 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$(\mathrm{ek}, c_0, \ldots, c_\ell)$
00 $x, r \xleftarrow{\$} \mathbb{Z}_p$	04 Parse ek as $(\mathrm{ek}_0, \mathrm{ek}_1)$	09 Parse c as (c_0, c_1)	12 Parse ek as $(\mathrm{ek}_0, \mathrm{ek}_1)$
01 $\mathrm{dk} \leftarrow x$	05 $s \xleftarrow{\$} \mathbb{Z}_p$	10 $m \leftarrow c_1 \cdot c_0^{-\mathrm{dk}}$	13 $r' \xleftarrow{\$} \mathbb{Z}_p$
02 $\mathrm{ek} \leftarrow (g^r, g^{xr})$	06 $c_0 \leftarrow \mathrm{ek}_0^s$	11 Return m	14 $\mathrm{ek}' \leftarrow (\mathrm{ek}_0^{r'}, \mathrm{ek}_1^{r'})$
03 Return $(\mathrm{ek}, \mathrm{dk})$	07 $c_1 \leftarrow \mathrm{ek}_1^s \cdot m$		15 For $i \in [\ell]$:
	08 Return (c_0, c_1)		16 Parse c_i as (c_i^0, c_i^1)
			17 $s'_i \xleftarrow{\$} \mathbb{Z}_p$
			18 $c'_i \leftarrow (c_i^0 \cdot \mathrm{ek}_0^{s'_i}, c_i^1 \cdot \mathrm{ek}_1^{s'_i})$
			19 Return $(\mathrm{ek}', c'_0, \ldots, c'_\ell)$

Fig. 3. Randomizable PKE scheme $\mathsf{rPKE}_{\mathsf{EG}}$.

Lemma 1. *Scheme* $\mathsf{rPKE}_{\mathsf{EG}}$ *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*

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

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:

- $\mathsf{ek} \xleftarrow{\$} \mathsf{urPKE.rr}(\mathsf{ek})$ outputs a randomized encryption key ek.
- $\mathsf{dk} \leftarrow \mathsf{urPKE.nextDk}(\mathsf{dk}, r)$ updates the decryption key with randomness r.
- $\mathsf{ek} \leftarrow \mathsf{urPKE.nextEk}(\mathsf{ek}, r)$ 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 $(\mathsf{ek}, \mathsf{dk})$ in the support of urPKE.gen, the two distributions $(\mathsf{urPKE.nextEk}(\mathsf{ek}, r), \mathsf{urPKE.nextDk}(\mathsf{dk}, r))$ and $(\mathsf{ek}', \mathsf{dk}') \xleftarrow{\$} \mathsf{urPKE.gen}$ are the same.

Indistinguishability of Randomizations. Similar to rPKE, we require for IND-R (formally defined in the full version [17]) security that an encryption key that has been randomized is indistinguishable from a freshly generated encryption key. In particular, the two distributions $(\mathsf{ek}, \mathsf{ek}_1)$ and $(\mathsf{ek}, \mathsf{ek}_2)$, where $(\mathsf{ek}, \_) \xleftarrow{\$} \mathsf{urPKE.gen}$, $\mathsf{ek}_1 \leftarrow \mathsf{urPKE.rr}(\mathsf{ek})$, $(\mathsf{ek}_2, \_) \xleftarrow{\$} \mathsf{urPKE.gen}$ 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 the full version [17].

Construction. We construct an updatable and randomizable PKE scheme based on hashed ElGamal, which was first proven to be IND-C secure in [1]. The construction is also similar to the secretly key-updatable encryption scheme of [26], thus we will only sketch it here. We give the full scheme in the full version [17], 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.

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 [22], H is a one way function.

Construction. Our construction of an updatable one-time signature scheme is given in Fig. 4. 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 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 [11, 12]. 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 the full version [17] we prove one-time existential unforgeability of the scheme.

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.

Correctness. We extend correctness requirements (1), (2) from the previous section by the following: We require that for all $(\mathrm{vk}, \mathrm{sk}) \in \mathsf{urSIG.gen}$, $m \in \mathcal{M}$,

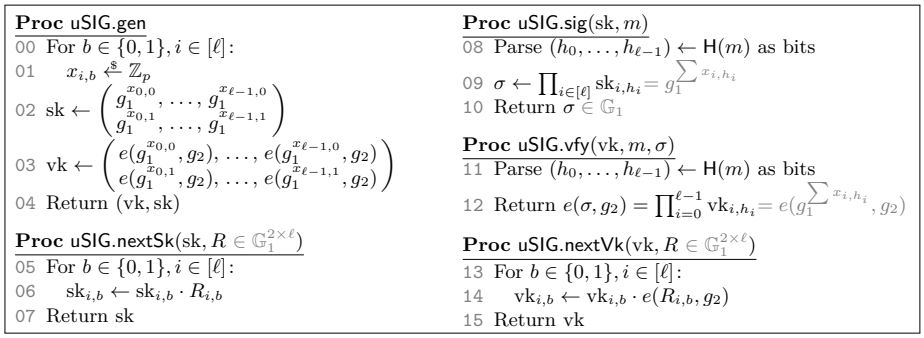

Fig. 4. Updatable one-time signature scheme uSIG for a pairing group $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where $\mathsf{H}\colon \{0,1\}^* \mapsto \{0,1\}^\ell$ is a hash function.

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.

In the full version [17] define additional security properties that are needed for authenticity and recover security.

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* 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. 5 we extend the updatable signature scheme in Fig. 4 by the randomizable PKE in Fig. 3 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 [13,29] using a strongly unforgeable signature.

Proc 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 $\quad (\mathrm{sk}_{i,b}^{(r)}, \mathrm{sk}_{i,b}^{(x)}) \leftarrow \mathsf{rPKE.enc}(\mathrm{ek}, m = \mathrm{sk}'_{i,b})$
04 $\mathrm{sk}^{(r)} \leftarrow \begin{pmatrix} \mathrm{sk}_{0,0}^{(r)}, \ldots, \mathrm{sk}_{\ell-1,0}^{(r)} \\ \mathrm{sk}_{0,1}^{(r)}, \ldots, \mathrm{sk}_{\ell-1,1}^{(r)} \end{pmatrix}$
$= \begin{pmatrix} g^{r_{0,0}}, \ldots, g^{r_{\ell-1,0}} \\ g^{r_{0,1}}, \ldots, g^{r_{\ell-1,1}} \end{pmatrix}$
05 $\mathrm{sk}^{(x)} \leftarrow \begin{pmatrix} \mathrm{sk}_{0,0}^{(x)}, \ldots, \mathrm{sk}_{\ell-1,0}^{(x)} \\ \mathrm{sk}_{0,1}^{(x)}, \ldots, \mathrm{sk}_{\ell-1,1}^{(x)} \end{pmatrix}$
$= \begin{pmatrix} g^{r_{0,0}\mathrm{dk}} g^{x_{0,0}}, \ldots, g^{r_{\ell-1,0}\mathrm{dk}} g^{x_{\ell-1,0}} \\ g^{r_{0,1}\mathrm{dk}} g^{x_{0,1}}, \ldots, g^{r_{\ell-1,1}\mathrm{dk}} g^{x_{\ell-1,1}} \end{pmatrix}$
06 $\mathrm{vk} \leftarrow (\mathrm{vk}', \mathrm{dk})$; $\mathrm{sk} \leftarrow (\mathrm{ek}, (\mathrm{sk}_{i,b}^{(r)}, \mathrm{sk}_{i,b}^{(x)}))$
07 Return $(\mathrm{vk}, \mathrm{sk})$

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

Proc urSIG.vfy(vk, m, σ)
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 urSIG.nextSk(sk, r)
15 Return $\mathsf{uSIG.nextSk}(\mathrm{sk}, r)$

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

Fig. 5. Our updatable and randomizable one-time signature scheme $\mathsf{urSIG}[\mathsf{rPKE}, \mathsf{uSIG}]$.

7 Construction of Anonymous RKE

Our construction of anonymous unidirectional RKE in Figure 6 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).

Proc 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 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 RKE.rr(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 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. 6. Construction of our RKE scheme $\mathsf{RKE}[\mathsf{urPKE}, \mathsf{urSIG}, \mathsf{H}, \mathsf{PRG}]$.

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 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 the full version [17]. 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 the full version [17].

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 the full version [17].

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 the full version [17].

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*

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

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

We provide a proof sketch below and defer the full proof to the full version [17].

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 $\texttt{ChallExpose}_R$, since, by definition of our trivial attack detection and instance independence, the adversary may call oracle $\texttt{ChallExpose}_R$ 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 $\texttt{ChallExpose}_S$ 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

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: CT-RSA 2001
2. Alwen, J., Auerbach, B., Noval, M.C., Klein, K., Pascual-Perez, G., Pietrzak, K., Walter, M.: CoCoA: Concurrent continuous group key agreement. In: EUROCRYPT 2022, Part II
3. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: EUROCRYPT 2019, Part I
4. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: CRYPTO 2020, Part I
5. Balli, F., Rösler, P., Vaudenay, S.: Determining the core primitive for optimally secure ratcheting. In: ASIACRYPT 2020, Part III
6. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-14, IETF
7. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: ASIACRYPT 2001
8. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. In: CRYPTO 2017, Part III
9. Bienstock, A., Dodis, Y., Garg, S., Grogan, G., Hajiabadi, M., Rösler, P.: On the worst-case inefficiency of CGKA. In: TCC 2022. LNCS
10. Bienstock, A., Dodis, Y., Rösler, P.: On the price of concurrency in group ratcheting protocols. In: TCC 2020, Part II
11. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: EUROCRYPT 2003
12. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: ASIACRYPT 2001
13. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: EUROCRYPT 2004
14. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In: ACM CCS 2018
15. Degabriele, J.P., Stam, M.: Untagging Tor: A formal treatment of onion encryption. In: EUROCRYPT 2018, Part III
16. Dodis, Y., Karthikeyan, H., Wichs, D.: Updatable public key encryption in the standard model. In: TCC 2021, Part III
17. Dowling, B., Hauck, E., Riepel, D., Rösler, P.: Strongly anonymous ratcheted key exchange. Cryptology ePrint Archive, Paper 2022/1187, https://eprint.iacr.org/2022/1187
18. Dowling, B., Rösler, P., Schwenk, J.: Flexible authenticated and confidential channel establishment (fACCE): Analyzing the noise protocol framework. In: PKC 2020, Part I
19. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. In: IWSEC 19
20. Emura, K., Kajita, K., Nojima, R., Ogawa, K., Ohtake, G.: Membership privacy for asynchronous group messaging. Cryptology ePrint Archive, Report 2022/046, https://eprint.iacr.org/2022/046
21. Fischlin, M.: Anonymous signatures made easy. In: PKC 2007

22. Galbraith, S.D., Hess, F., Vercauteren, F.: Aspects of pairing inversion. Cryptology ePrint Archive, Report 2007/256, https://eprint.iacr.org/2007/256
23. Grubbs, P., Maram, V., Paterson, K.G.: Anonymous, robust post-quantum public key encryption. In: EUROCRYPT 2022, Part III
24. Ishibashi, R., Yoneyama, K.: Post-quantum anonymous one-sided authenticated key exchange without random oracles. In: PKC 2022
25. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: The safety of messaging. In: CRYPTO 2018, Part I
26. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: Almost-optimal guarantees for secure messaging. In: EUROCRYPT 2019, Part I
27. Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: Anonymity-preserving public-key encryption: A constructive approach. In: PETS 2013
28. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory
29. MacKenzie, P.D., Reiter, M.K., Yang, K.: Alternatives to non-malleability: Definitions, constructions, and applications (extended abstract). In: TCC 2004
30. Martiny, I., Kaptchuk, G., Aviv, A.J., Roche, D.S., Wustrow, E.: Improving signal's sealed sender. In: NDSS 2021
31. Perrin, T.: The noise protocol framework. http://noiseprotocol.org/noise.html, revision 34
32. Perrin, T., Marlinspike, M.: The double ratchet algorithm. https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf
33. Poettering, B., Rösler, P.: Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296, https://eprint.iacr.org/2018/296
34. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: CRYPTO 2018, Part I
35. Rogaway, P., Zhang, Y.: Onion-ae: Foundations of nested encryption. Proc. Priv. Enhancing Technol.
36. Rogaway, P., Zhang, Y.: Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In: CRYPTO 2018, Part II
37. Rösler, P., Mainka, C., Schwenk, J.: More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema. In: IEEE EuroS&P 2018
38. Schäge, S., Schwenk, J., Lauer, S.: Privacy-preserving authenticated key exchange and the case of IKEv2. In: PKC 2020, Part II
39. Signal: Sealed sender. https://signal.org/blog/sealed-sender/, blog post
40. Tyagi, N., Len, J., Miers, I., Ristenpart, T.: Orca: Blocklisting in sender-anonymous messaging. In: USENIX Security 2022
41. Yang, G., Wong, D.S., Deng, X., Wang, H.: Anonymous signature schemes. In: PKC 2006
42. Zhao, Y.: Identity-concealed authenticated encryption and key exchange. In: ACM CCS 2016