# A More Complete Analysis of the Signal Double Ratchet Algorithm*

Alexander Bienstock[1], Jaiden Fairoze[2], Sanjam Garg[2,3], Pratyay Mukherjee[4], and Srinivasan Raghuraman[5]

[1] New York University
[2] UC Berkeley
[3] NTT Research
[4] Swirlds Labs
[5] Visa Research

**Abstract.** Seminal works by Cohn-Gordon, Cremers, Dowling, Garratt, and Stebila [EuroS&P 2017] and Alwen, Coretti and Dodis [EUROCRYPT 2019] provided the first formal frameworks for studying the widely-used Signal Double Ratchet (DR for short) algorithm.

In this work, we develop a new Universally Composable (UC) definition $\mathcal{F}_{\mathsf{DR}}$ that we show is provably achieved by the DR protocol. Our definition captures not only the security and correctness guarantees of the DR already identified in the prior state-of-the-art analyses of Cohn-Gordon *et al.* and Alwen *et al.*, but also *more* guarantees that are absent from one or *both* of these works. In particular, we construct *six* different modified versions of the DR protocol, all of which are insecure according to our definition $\mathcal{F}_{\mathsf{DR}}$, but remain secure according to one (or both) of their definitions. For example, our definition is the first to fully capture CCA-style attacks possible immediately after a compromise — attacks that, as we show, the DR protocol provably resists, but were not fully captured by prior definitions.

We additionally show that multiple compromises of a party in a short time interval, which the DR is expected to be able to withstand, as we understand from its whitepaper, nonetheless introduce a new non-trivial (albeit minor) weakness of the DR. Since the definitions in the literature (including our $\mathcal{F}_{\mathsf{DR}}$ above) do not capture security against this more nuanced scenario, we define a new stronger definition $\mathcal{F}_{\mathsf{TR}}$ that does.

Finally, we provide a *minimalistic modification* to the DR (that we call the Triple Ratchet, or TR for short) and show that the resulting protocol securely realizes the stronger functionality $\mathcal{F}_{\mathsf{TR}}$. Remarkably, the modification incurs no additional communication cost and virtually no additional computational cost. We also show that these techniques can be used to improve communication costs in other scenarios, e.g. practical Updatable Public Key Encryption schemes and the re-randomized TreeKEM protocol of Alwen *et al.* [CRYPTO 2020] for Secure Group Messaging.

* The full version [8] is available as entry 2022/355 in the IACR eprint archive.

## 1 Introduction

*Background.* The Signal protocol is by far the most popular end-to-end secure messaging (SM) protocol, boasting of billions of users. Based on the Off-The-Record protocol [10], the core underlying technique of the Signal protocol is commonly known as the *Double Ratchet* (DR) algorithm. The DR is beautifully explained in the whitepaper [40] authored by the creators of Signal, Marlinspike and Perrin. The whitepaper also outlines the desired security properties of the DR, and provides intuition on the design rationale for achieving them. Indeed, in addition to standard security against an eavesdropper who may modify ciphertexts, the DR attempts to achieve (i) *post-compromise security* (PCS) and *forward secrecy* (FS) with respect to leakages of secret state, (ii) *resilience to bad randomness*, and (iii) *immediate decryption* (all at the same time). PCS requires the conversation to naturally and quickly recover security after a leakage on one of the (or both) parties, as long as the affected parties have good randomness (and the adversary remains passive while such recovery occurs) [20]. FS requires past messages to remain secure even after a leakage on one of the (or both) parties. Resilience to bad randomness requires that as long as both parties' secret states are secure (i.e., PCS has been achieved after any corruptions), then the conversation should remain secure, even if bad randomness is used in crafting messages. Finally, immediate decryption requires parties to — immediately upon reception of ciphertexts — obtain underlying plaintexts and place them in the correct order in the conversation, even if they arrive arbitrarily out of order and if some of them are completely lost by the network (the latter is also known as *message-loss resilience*).

However, despite the elegance and simplicity of the Double Ratchet, capturing its security turned out to be not so straightforward. The first formal analysis of the DR protocol (in fact, the whole Signal protocol) was provided by Cohn-Gordon *et al.* in EuroS&P 2017 [18,19] (referred to as CCD$^+$ henceforth). However, this analysis left open several questions about the cryptographic security and correctness achieved by the DR. Following in the footsteps of CCD$^+$, a more generic and comprehensive security definition of the DR was provided by Alwen *et al.* in Eurocrypt 2019 [3] (referred to as ACD henceforth), with close focus on the immediate decryption property of the DR protocol. They provided a modular analysis with respect to game-based definitions proposed therein. Indeed, they introduced new abstract primitives and composed them into SM protocols (including the DR itself) that capture the above properties: Continuous Key Agreement (CKA), Forward-Secure Authenticated Encryption with Associated Data (FS-AEAD), and PRF-PRNGs. While the works of CCD$^+$ and ACD significantly improved our understanding of the DR, as we observe in this work, both definitional frameworks do not capture some of its security and functionality properties.

### 1.1 Our Contributions

In this work, our key aim is to develop a formal definitional framework that captures the security and correctness properties of the DR protocol as completely as possible. Moreover, we aspire for definitions that are simple to state and easy to build on (e.g., imagine executing a Private Set Intersection Protocol on top of the DR). More specifically:

- **New Definitional Framework for the DR:** We provide a new definition $\mathcal{F}_{\mathsf{DR}}$ for the DR protocol, in the Universal Composability [14] (UC) framework. Our definition captures all of the security and correctness guarantees of the DR provided by ACD's and CCD$^+$'s definitions, but also *more* guarantees that are absent from one or *both* of these works. To demonstrate this, we construct *six* different (albeit somewhat contrived) modified versions of the DR protocol, all of which are insecure according to our definition, but remain secure according to ACD's and/or CCD$^+$'s definition. Some of these transformations are indeed based on analyzed (weaker) DR variants in the literature, while others are based on novel observations. For example, our definition is the first to fully capture CCA-style attacks that become possible on the DR immediately after a party has been compromised — attacks that, as we show, the DR provably resists, but were not fully captured by prior definitions. We provide an overview of our new definition's advantages in Section 1.3.

  Finally, we prove that the DR protocol, as it is described in the whitepaper [40] (in its strongest form), securely realizes our ideal functionality $\mathcal{F}_{\mathsf{DR}}$. Our proof is modular and proceeds by expanding on ACD's modular definitional framework (see Appendix E). Note that we model part of the underlying AEAD of the DR using a programmable ideal cipher to prove security in the UC setting where an adversary can corrupt a party while a (heretofore secure) ciphertext is in transit.
- **Non-trivial (albeit minor) weakness of the DR:** We find that multiple compromises of a party in a short time interval, which the DR should be able to withstand, as we understand from its whitepaper, nonetheless introduce a new non-trivial (albeit minor) weakness of the DR. This weakness is allowed in the definitions of both ACD and CCD$^+$, as well as $\mathcal{F}_{\mathsf{DR}}$, so we provide a new stronger definition $\mathcal{F}_{\mathsf{TR}}$ that does not allow it. We summarize this compromise scenario in Section 1.4.
- **Achieving stronger security:** Finally, we complement the above weakness by providing a minimalistic modification to the DR and prove the resulting protocol secure according to the stronger definition $\mathcal{F}_{\mathsf{TR}}$. We call this new protocol the Triple Ratchet (TR) as it adds another "mini ratchet" to the public ratchet in the DR Protocol. Remarkably, the modification incurs no additional communication cost and virtually no additional computational cost. We provide an overview of the TR in Section 1.5.

  We believe that the techniques realized here are also likely to find other applications. For instance, in Appendix F, we show that our techniques

can be used to improve current practical Updatable Public Key Encryption (UPKE) constructions [4,34], reducing their communication cost by an additive factor of $|G|$, where $|G|$ is the number of bits needed to represent the size of the (CDH-hard) group used in the construction, without any additional computational cost. Furthermore, the technique yields an improvement to the communication cost of the re-randomized TreeKEM (rTreeKEM) protocol of Alwen *et al.* [4] — specifically, improving the communication cost by up to roughly an additive factor of $|G| \cdot n$, where $n$ is the number of users in the group.

### 1.2 High-Level Summary of the Double Ratchet and its Security Properties

Before elaborating on our results in the subsequent sections, we first give a high-level overview of the Signal Double Ratchet and its security properties which we capture in our definition. For another detailed description we refer to the Double Ratchet whitepaper [40]. Readers familiar with the Double Ratchet algorithm could easily skip this section.

We note that although we here describe the double ratchet specifically in terms of its real-world implementation [40], our paper still breaks it down into modular pieces which can be instantiated in several different ways, as in ACD. For the purpose of our paper, we assume that the two participants $\mathsf{P}_1$ and $\mathsf{P}_2$ share a common secret upon initialization. In Signal, this is achieved via the X3DH key exchange protocol [41], but we consider this out of scope for our study of the double ratchet. Using their initial shared secret, $\mathsf{P}_1$ and $\mathsf{P}_2$ can derive the initial *root key* $\sigma$ which seeds the public ratchet. Furthermore, upon initialization $\mathsf{P}_2$ also holds some secret exponent $x_0$ and $\mathsf{P}_1$ holds the corresponding public value $g^{x_0}$. Once the initialization process completes, the ratcheting session begins.

At its core, the double ratchet has two key components: the outer public-key ratchet, and the inner symmetric-key ratchet (often referred to as simply the public and symmetric ratchets, respectively). ACD abstract out the symmetric ratchet as their FS-AEAD primitive, the update mechanism of the public ratchet as their PRF-PRNG primitive, and the means by which shared secrets are produced to update the public ratchet as their CKA primitive. The goal of the double ratchet is to provide distinct *message keys* to encrypt/decrypt each new message. For each message the same message key is derived by both parties using a symmetric *chain key* which itself is derived from the aforementioned root key. Naturally, this results in a key *hierarchy* with the root key at the top, chain keys at an intermediate layer, and message keys at the bottom. Observe a graphical depiction of this hierarchy in Figure 1. In the Signal double ratchet, Diffie-Hellman key exchange is used to "ratchet forward" the root key, which can then be used to establish corresponding symmetric chain keys. Message keys are then derived from the current (newest) chain key, where chain keys are updated deterministically such that multiple messages can be sent in a row before

a response, and no matter which of these messages is the first to arrive, the recipient can always compute its corresponding message key *immediately.* We now introduce the concept of asynchronous epochs before describing the two ratchets and the primary properties which they achieve:

*Asynchronous Sending Epochs.* In the double ratchet, the parties $\mathsf{P}_1$ and $\mathsf{P}_2$ asynchronously alternate sending messages in *epochs* (as termed in [3]): Assume that $\mathsf{P}_1$ starts the conversation, sending in epoch 1 at least one message. Then once $\mathsf{P}_2$ receives one of these messages, she sends messages in epoch 2. Furthermore, once $\mathsf{P}_1$ receives one of these message, she starts epoch 3, and so on. We emphasize that these sending epochs are *asynchronous* – for example, even if $\mathsf{P}_2$ has started sending in epoch 2, if $\mathsf{P}_1$ has not yet received any such epoch 2 messages and wants to send new messages, she will still send them in epoch 1. Not until she finally receives one of $\mathsf{P}_2$'s epoch 2 messages will she send new messages in epoch 3.

*Public Ratchet.* The public ratchet forms the backbone of the double ratchet algorithm. Parties update the root key using public-key cryptography (i.e. Diffie-Hellman secrets) every time a new epoch is initiated: if $\mathsf{P}_1$ wishes to start a new epoch, she must first update the root key using the Diffie-Hellman public value from $\mathsf{P}_2$'s latest epoch (or initialization). After deriving a new chain key from the root key, $\mathsf{P}_1$ can send multiple separate messages in a row—this involves deriving a new message key for each message via the symmetric ratchet, as explained below.

We now describe the root key update process in more detail. To start a new epoch $t$, $\mathsf{P}_1$ samples a new private exponent $x_t$ and corresponding public value $g^{x_t}$. Next, she uses the public value received from $\mathsf{P}_2$'s latest epoch (or initialization), say $g^{x_{t-1}}$, to compute a shared secret $(g^{x_{t-1}})^{x_t} = g^{x_{t-1}x_t}$. Then, $\mathsf{P}_1$ uses a two-input Key Derivation Function (KDF) to update the current root key and derive a new chain key in one go. That is, she computes $(\sigma_t, w_{t,1}) \leftarrow \mathrm{KDF}(\sigma_{t-1}, g^{x_{t-1}x_t})$. Observe that even if $\mathsf{P}_1$'s state was leaked before this update, as long as the parties used good randomness in sampling their Diffie-Hellman keys, the new root key and chain key will be secure. This is the key to achieving PCS. Symmetrically, even if $\mathsf{P}_1$ uses bad randomness when performing this update, as long as if $\sigma_{t-1}$ was secure, then the new root key and chain key will be secure. Furthermore, root keys are clearly forward secret, from the security of the KDF and the fact that new Diffie-Hellman secrets are sampled independently of past ones.

$\mathsf{P}_1$ includes in every message of the new epoch the fresh public share $g^{x_t}$ to allow $\mathsf{P}_2$ to compute the new shared secret $g^{x_{t-1}x_t}$ that is used to update the root key, no matter which message of the epoch she receives first. This in part is what provides for immediate decryption (and message loss resilience). When $\mathsf{P}_2$ receives a message in $\mathsf{P}_1$'s new epoch, she recomputes the same above steps, i.e. she computes $\sigma_t$ by first computing $(g^{x_t})^{x_{t-1}} = g^{x_{t-1}x_t}$ where $x_{t-1}$ is $\mathsf{P}_2$'s own private share, followed by the same KDF computation. Once $\mathsf{P}_2$ wishes to start her own new epoch, she generates another Diffie-Hellman pair $(x_{t+1}, g^{x_{t+1}})$ to

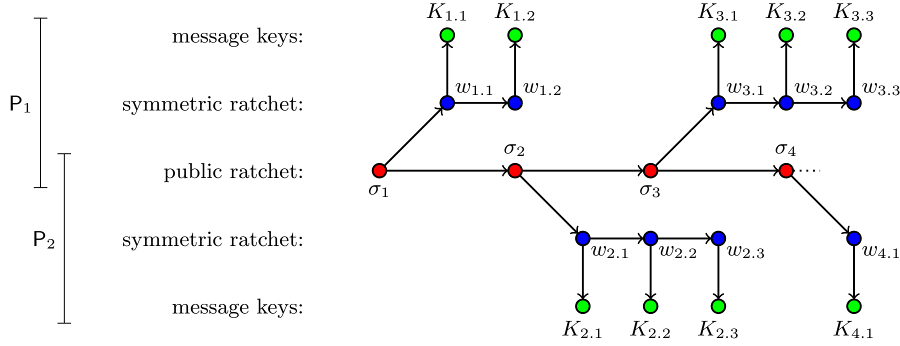



Fig. 1: Sample Double Ratchet key evolution. In this depiction, $\mathsf{P}_1$ sends and $\mathsf{P}_2$ receives in epoch 1, followed by $\mathsf{P}_2$ sending and $\mathsf{P}_1$ receiving in epoch 2, and so on. As explained in the main body, initial symmetric chain keys $w_{i,1}$ for each epoch $i$ are derived first by the sender, then also by the receiver, using the shared root keys $\sigma_i$ and asynchronously exchanged shared secrets (via DDH). Then, updated symmetric chain keys $w_{i,j}$ and message keys $K_{i,j}$ are derived deterministically from $w_{i,1}$.

ratchet the root key forward $(\sigma_{t+1}, w_{t+1,1}) \leftarrow \mathrm{KDF}(\sigma_t, g^{x_t x_{t+1}})$. Essentially, $\mathsf{P}_2$ has refreshed her component of the Diffie-Hellman shared secret while reusing $\mathsf{P}_1$'s value from the previous epoch. Now, when $\mathsf{P}_1$ receives a message for this epoch and again wishes to start a new one, she would similarly need to sample a new Diffie-Hellman share $x_{t+2}$. This process can continue asynchronously for as long as the session is active.

*Symmetric Ratchet.* The main purpose of the symmetric ratchet is to produce single-use symmetric keys for message encryption. When a party wishes to send (or receive) the next ($i$th) message in some epoch $t$, they derive a distinct message key $K_{t,i}$ from the symmetric chain key $w_{t,i}$ and simultaneously update the chain key. This is done by applying a KDF as follows: $(w_{t,i+1}, K_{t,i}) \leftarrow \mathrm{KDF}(w_{t,i})$ (if the KDF requires two inputs, a fixed value may be used to fill the other input). Observe that the symmetric ratchet is clearly forward secret from the security of the KDF. Note however that the symmetric ratchet does not have PCS due to its deterministic nature.

So, if $\mathsf{P}_1$ just started a new epoch then she first computes initial symmetric chain key $w_{t,1}$ for the epoch as above. To derive a message key, $\mathsf{P}_1$ puts this new chain key through the KDF to compute $(w_{t,2}, K_{t,1}) \leftarrow \mathrm{KDF}(w_{t,1})$. If $\mathsf{P}_1$ wishes to send a second message, then she can derive $(w_{t,3}, K_{t,2}) \leftarrow \mathrm{KDF}(w_{t,2})$. When $\mathsf{P}_2$ receives these messages from $\mathsf{P}_1$, she can repeat the key derivation in the same way as $\mathsf{P}_1$ and use the subsequent message keys to decrypt the messages, no matter the order in which they arrive. The deterministic nature of the symmetric ratchet, along with including the public ratchet values in every message as above, provides immediate decryption.

### 1.3 High-Level Summary of our DR Definition's Strength over Prior Notions

In Section 2 we fully formalize our new definition for the DR in the UC framework, $\mathcal{F}_{\mathsf{DR}}$, and provide thorough discussion on it. Then in Appendix E, we show that the DR UC-realizes $\mathcal{F}_{\mathsf{DR}}$ (in the programmable ideal cipher model). Intuitively, $\mathcal{F}_{\mathsf{DR}}$ captures all of the properties described in the last section, including all of those captured by the definitions of ACD and CCD$^+$. Here we emphasize several properties which $\mathcal{F}_{\mathsf{DR}}$ guarantees, but at least one of ACD's and CCD$^+$'s definitions do not. We do so by providing *six* distinct transformations to the original DR protocol (denoted as $T_i(\mathsf{DR})$ for $i \in [6]$), showing their natural vulnerabilities here, and their insecurity according to $\mathcal{F}_{\mathsf{DR}}$, but security according to at least one of ACD's (transformations $1-4$) and CCD$^+$'s (transformations $4-6$) definitions. We show this formally in Appendix D.2. Although some of these transformations may be seen as artificial, they emphasize that our definition is stronger than those of ACD and CCD$^+$. Below we use the formalization of symmetric and public ratchets as done by ACD and also adapted by us – the symmetric ratchet is abstracted out as an *FS-AEAD* scheme and the public ratchet as a *CKA* scheme. We defer these definitions to Section 3 and Appendix B.

$T_1$*: Postponed* FS-AEAD *Key Deletion:* This transformation slightly modifies the handling of symmetric ratchet secrets. In particular, when a party receives a new message for its counterpart's next epoch, it does not immediately delete its (no longer needed) symmetric ratchet secrets from its previous sending epoch. Instead, it waits to delete these secrets until it starts its next sending epoch (i.e., sends its next message). In that case, an injection attack can be launched as follows: only focusing on the symmetric ratchet, suppose that for a sending epoch $t$, $\mathsf{P}_1$ derives $(w_{t,2}, K_{t,1}) \leftarrow \mathrm{KDF}(w_{t,1})$ and sends an encrypted message using $K_{t,1}$, that is then received by $\mathsf{P}_2$. Then $\mathsf{P}_2$ sends a message in epoch $t+1$, which is received by $\mathsf{P}_1$. Observe that unlike in (the strongest version of) the DR, $T_1(\mathsf{DR})$ keeps $w_{t,2}$ in $\mathsf{P}_1$'s memory even after receiving this epoch $t+1$ message from $\mathsf{P}_2$. Now if $\mathsf{P}_1$ is compromised then the attacker obtains $w_{t,2}$. Using this it can now launch an injection attack for $\mathsf{P}_1$'s sending epoch $t$ (not just $\mathsf{P}_1$'s next sending epoch, $t+2$) by encrypting any arbitrary message of its choice using the next message key $(\cdot, K_{t,2}) \leftarrow \mathrm{KDF}(w_{t,2})$ and sending that to $\mathsf{P}_2$. Note that each time a sending epoch is started in the protocol, the information about how many messages were sent in the immediately past sending epoch is included. Nonetheless, that does not thwart this attack, because it is launched even before $\mathsf{P}_1$ starts the next sending epoch.

Although this transformation is perhaps artificial, one can imagine scenarios in which the relative timing of messages sent by the two parties is important. Perhaps more importantly, it is clearly less secure than the standard (most secure version of) DR, but, remarkably, the version described by ACD is indeed $T_1(\mathsf{DR})$. Furthermore, as evident by ACD's security proof, their definition therefore does

not require resistance against this attack; intuitively making our (and CCD$^+$'s) definition stronger than theirs in this respect.

*$T_2$: Postponed* CKA *Key Deletion:* A similar problem arises if the keys from the public ratchet are kept for too long. The transformed protocol works as follows: suppose that in starting a new sending epoch $t$, $\mathsf{P}_2$ samples a secret exponent $x_t$ and combines it with the public ratchet message of $\mathsf{P}_1$'s prior sending epoch, $g^{x_{t-1}}$, to compute $I_t = g^{x_{t-1}x_t}$. Then, $\mathsf{P}_2$ proceeds to send several messages using $I_t$ (and the root key for the KDF, as described in Section 1.2) as normal. When receiving a message for the first time in sending epoch $t$ of $\mathsf{P}_2$, $\mathsf{P}_1$ uses her stored secret exponent $x_{t-1}$ and combines it with $\mathsf{P}_2$'s public ratchet message $g^{x_t}$ to compute $I_t$. However, at this point, instead of deleting $I_t$ (as done in the normal DR protocol), $\mathsf{P}_1$ saves it in $T_2(\mathsf{DR})$. Now assume that $\mathsf{P}_1$ receives all of $\mathsf{P}_2$'s epoch $t$ messages. Then, when $\mathsf{P}_1$ again switches to a new sending epoch she generates a new $I_{t+1}$ (deleting the old $I_t$). An attack can be executed on $T_2(\mathsf{DR})$, by simply corrupting $\mathsf{P}_1$ before the start of epoch $t+1$, and then using the leaked $I_t$ to decrypt the already delivered messages sent by $\mathsf{P}_2$ in epoch $t$ – thus breaking forward security. Note: this also requires another corruption of $\mathsf{P}_2$ *before* she sends messages in the attacked epoch $t$, to obtain the root key for the KDF. ACD's definition explicitly prevents querying the challenge oracle immediately after corruptions, and thus does not require resistance against this attack. CCD$^+$'s model does explicitly require resistance against this attack. This transformation may seem artificial, but clearly allowing the adversary to decrypt old messages should not be allowed in any formal model of the DR, and in fact is not allowed in $\mathcal{F}_{\mathsf{DR}}$.

*$T_3$: Eager* CKA *Randomness Sampling:* If the secret-exponent of a public-ratchet is sampled too early, then that makes the protocol vulnerable. For example, consider $T_3(\mathsf{DR})$ in which $\mathsf{P}_1$ samples the exponent $x_t$ for the next sending epoch $t$ when still in receiving epoch $t-1$. An attacker may compromise $\mathsf{P}_1$ to obtain $x_t$ (and the root key) at this stage and use that to decrypt the messages sent in the next epoch $t$, thereby breaking PCS. ACD's definition does not require resistance against this attack, while $\mathcal{F}_{\mathsf{DR}}$ does, because their definition does not allow querying the challenge oracle immediately after corruptions. It is worth pointing out that the Double Ratchet whitepaper [40] and CCD$^+$ present $T_3(\mathsf{DR})$ and its early sampling as their primary description of the DR, though the whitepaper later suggests deferring randomness sampling until actually sending for better security, which we choose to model. However, the security model itself of CCD$^+$ only analyzes the key exchange component of the DR and we believe that it *could* indeed be composed with an AEAD scheme to avoid the weakness of $T_3(\mathsf{DR})$. However, this needs to be carefully done, and not according to their description of the full DR protocol.

*$T_4$: Malleable Ciphertexts:* If the protocol does not provide a strong non-malleability guarantee, then the DR protocol could suffer from a mauling attack according to our weaker definition $\mathcal{F}_{\mathsf{DR}}$. More specifically, if the root key is leaked, and $T_4(\mathsf{DR})$

uses a weak mechanism to update the public ratchet (note: the DR public ratchet should provide PCS here), there may exist attacks which, for example, can successfully maul DR ciphertexts encrypting $m$ into new ones that decrypt to $m+1$. This becomes evident when we prove the DR protocol secure according to $\mathcal{F}_{\mathsf{DR}}$, which is required to protect against such an attack, as we need to rely on such a non-malleability property. Indeed, the DR seems to require modelling the public ratchet KDF as a random oracle and that the Strong Diffie-Hellman assumption (StDH) is secure (i.e., given random and independent $g^a, g^b$, and oracle access to $\mathbf{ddh}(g^a, \cdot, \cdot)$ which on input group elements $X, Y$ checks if $X^a = Y$, it is hard to compute $g^{ab}$), in order to realize $\mathcal{F}_{\mathsf{DR}}$. To provide evidence for this requirement: the ciphertexts and key material known to the adversary in the above scenario are almost identical to that of Hashed ElGamal encryption, for which all analyses of its CCA-security of which we are aware use the same assumptions [1, 21, 39]. We do not rule out a security proof from weaker assumptions, however, it seems that using a group in which only DDH is hard, and not, for example, StDH, for the public ratchet could lead to an attack like the above. ACD's definition does not require resistance against such an attack since it does not allow injections after corruptions; therefore, their security proof only relies on DDH.

CCD$^+$'s definition also does not completely cover mauling attacks, since it only analyzes the key exchange component of the DR, not any actual message transmission. Therefore, if one composes a key exchange protocol secure with respect to CCD$^+$'s definition with a non-authenticated encryption scheme, it would not provide the non-malleability guarantees required by $\mathcal{F}_{\mathsf{DR}}$.

*$T_5$: CKA Bad Randomness Plaintext Trigger:* The DR is very resilient to attacks against its source of randomness. However, in $T_5(\mathsf{DR})$, if a party samples a certain string of random bits, say the all-0 string, then it (rather artificially) sends the rest of its messages in the conversation as plaintext. In our (and ACD's) model, which require security even if the adversary can supply the parties with random bits each time they attempt to sample randomness, such a protocol is clearly insecure. However, CCD$^+$'s model only allows randomness *reveals* of *uniformly* sampled random bits. Thus sampling the all-0 string occurs with negligible probability (if we assume bit strings of poly($\lambda$) length), so security in CCD$^+$'s model is retained. Although this attack is quite artificial, [5] note that attacks on randomness sources (e.g., [31]) and/or generators (e.g., [17, 51]) are not captured by randomness reveals, but are captured by randomness manipulation as in our model. Furthermore, [5] show that including randomness manipulations has a concrete effect on protocol construction, particularly in Secure Messaging.

*$T_6$: Removed Immediate Decryption:* Finally, $T_6(\mathsf{DR})$ changes the DR to include the public ratchet message as part of only the first ciphertext of an epoch. It is thus pretty simple to violate the immediate decryption property required by our ideal functionality: First have $\mathsf{P}_1$ send two messages $m_1, m_2$ in a new epoch $t$, generating ciphertexts $c_1$ and $c_2$. Then, attempt to deliver $c_2$ to $\mathsf{P}_2$ (before $c_1$). Since $c_2$ does not include the public ratchet message of the epoch, $\mathsf{P}_2$ will be unable to decrypt it to obtain $m_2$. While $\mathcal{F}_{\mathsf{DR}}$ does in fact require immediate

decryption, CCD$^+$'s model does not require it (nor correctness more generally), so $T_6(\mathsf{DR})$ satisfies all formal requirements of their model. ACD's model does in fact require immediate decryption.

Although this too may be an artificial transformation, immediate decryption is an important practical property of the DR, and one of the DR's main novelties is obtaining immediate decryption at the same time as FS and PCS. Furthermore, properly modelling immediate decryption allows subsequent work to understand it, and further improve upon the DR with the requirement in mind. Indeed, many of the works which we are aware of [7, 24, 32, 34, 45], besides [3], which try to improve the DR do not consider immediate decryption in their security models or constructions, arguably thrusting these works outside of the practical realm.

### 1.4 High-Level Summary of the DR's Minor Weakness

Here we show a scenario that introduces a new non-trivial (albeit minor) weakness of the DR which demonstrates a gap between the security guarantees that the DR should achieve according to our understanding of its whitepaper, and those which it actually does achieve. The attack utilizes two compromises of a party in a short time interval, and stems from the fact that a party needs to hold on to the secret exponent $x_t$ for the public ratchet that it generates in a sending epoch $t$ until it receives a message from its counterpart's next sending epoch $t+1$. Indeed secret exponent $x_t$ is needed until this point because the other party uses its public component to encrypt messages in epoch $t+1$. For example, consider a setting in that party $\mathsf{P}_1$ is about to start a sending epoch $t$. At this point $\mathsf{P}_1$'s state has $g^{x_{t-1}}$ and $\mathsf{P}_2$ has $x_{t-1}$. Now when the sending epoch commences, $\mathsf{P}_1$ samples fresh secret (random) exponent $x_t$ and combines that with $g^{x_{t-1}}$ to derive the CKA key $I_t = g^{x_{t-1}x_t}$, which she then combines with the root key $\sigma_t$ to derive first the symmetric chain key $w_{t,1}$, followed by message key $K_{t,1}$. Note that if $x_t$ is truly random, then $I_t$ and thus $w_{t,1}$ and all subsequent message keys $K_{t,i}$ should be secure. In this epoch $\mathsf{P}_1$ sends $g^{x_t}$ to $\mathsf{P}_2$, who then derives the same key $I_t$ by computing $(g^{x_t})^{x_{t-1}}$, and subsequently $K_{t,1}$. In the next epoch, $\mathsf{P}_2$ becomes a sender. Then $\mathsf{P}_2$ samples a fresh $x_{t+1}$ to derive a new CKA key $I_{t+1} = (g^{x_t})^{x_{t+1}}$ and sends $g^{x_{t+1}}$ to $\mathsf{P}_1$. Now, $\mathsf{P}_1$ needs to compute $I_{t+1}$ as $(g^{x_{t+1}})^{x_t}$. To execute this step $\mathsf{P}_1$ must have stored $x_t$ throughout its sending epoch. The attack exploits this by compromising $\mathsf{P}_1$ twice in a short interval:

- first compromise $\mathsf{P}_1$ before starting the sending epoch $t$ to obtain the root key $\sigma_t$;
- then compromise $\mathsf{P}_1$ at any time after she sends a few messages (at least one), but before she receives any epoch $t+1$ messages, to obtain $x_t$, and thus $I_t$;

and then combine $\sigma_t$ and $I_t$ to derive $w_{t,1}$, given which *all* messages within $\mathsf{P}_1$'s sending epoch $t$ are vulnerable, including the ones that were sent between the two corruptions. Intuitively, this breaks PCS with respect to the first corruption,

since as noted above, if $x_t$ is truly random, then the corresponding message keys should be secure, as well as FS with respect to the second corruption. For more details we refer to Appendix C.2.

In Section 2, we provide a new ideal functionality, $\mathcal{F}_{\mathsf{TR}}$, that strengthens $\mathcal{F}_{\mathsf{DR}}$ in order to capture security against the above compromise scenario. We note that both the definitions of ACD and CCD$^+$ also did not capture this scenario.

### 1.5 High-Level Summary of the Triple Ratchet

Finally, we provide a minimalistic modification of the DR, which we call the Triple Ratchet protocol, or simply TR, with virtually no overhead over the DR. This protocol is secure against the compromise scenario provided in the previous section and thus realizes our stronger ideal functionality, $\mathcal{F}_{\mathsf{TR}}$. The TR protocol modifies the underlying public ratchet in a way that the sampled secret exponent is deterministically updated after starting a sending epoch; thus, adding a "mini ratchet" on top of Signal's public ratchet. In particular, using the notation from above, in the modified public ratchet, a party (say $\mathsf{P}_1$) after sampling secret exponent $x_t$, and deriving $I_t = (g^{x_{t-1}})^{x_t}$, sends $g^{x_t}$ as the public ratchet message, but stores $x'_t = x_t \cdot \mathsf{H}(I_t)$ instead of $x_t$. Once $\mathsf{P}_2$ receives $g^{x_t}$, she also derives $I_t$ and computes $g^{x'_t} = g^{x_t \cdot \mathsf{H}(I_t)}$ that she uses for the next public ratchet. In particular, in the next epoch when $\mathsf{P}_2$ becomes the sender, she samples a fresh secret exponent $x_{t+1}$, and uses the key $I_{t+1} = g^{x'_t x_{t+1}}$. $\mathsf{P}_2$ sends $g^{x_{t+1}}$, upon receiving which $\mathsf{P}_1$ can compute $I_{t+1}$ as $(g^{x_{t+1}})^{x'_t}$, but $\mathsf{P}_2$ only stores $x'_{t+1} = x_{t+1} \cdot \mathsf{H}(I_{t+1})$, and so on. Assuming $\mathsf{H}$ to be a random oracle, or instead, circular-security of ElGamal encryption, we can show that given $x'_t$, $I_t$ is completely hidden, rendering the attack of the previous section useless. Note that the communication cost remains the same for the modified protocol, that is one group element. The computation cost increases only slightly, specifically exactly once per epoch. We also note that, the alternate CKA scheme based on generic KEMs proposed by [3] seems to achieve this security too, albeit with doubling the communication cost.

Furthermore, as we show in Appendix F, our efficient modification can also be applied to practical UPKE schemes, reducing their communication by an additive factor of $|G|$, where $|G|$ is the number of bits needed to represent the size of the (CDH-hard) group used in the schemes. Using the modified UPKE scheme, we can reduce the communication of, e.g., the rTreeKEM scheme [4] used for Secure Group Messaging by an additive factor of $|G| \cdot n$, where $n$ is the number of users in the group.

### 1.6 Other Related Work

Canetti, Jain, Swanberg, and Varia [16] also recently studied the security of the Signal protocol in the UC framework. Kobeissi, Bhargavan, and Blanchet [37] use automated verification tools to provide symbolic and computational proofs for a simplified variant of the Signal protocol.

Following the first formal analysis of Signal by CCD$^+$, researchers proposed a number of protocols that provided stronger security than the DR [5, 7, 25, 32, 34, 45]. ACD however observed that in the process of strengthening security, all such protocols suffer from steep efficiency costs and loss of *immediate decryption*, rendering these protocols impractical for real-world use.

Jost, Maurer and Mularczyk [35] analyzed ratcheting with the Constructive Cryptography framework [42]. They aimed to capture the security and composability of various sub-protocols, such as FS-AEAD, used in the construction of larger ratcheting protocols.

More recently, there has also been work on the X3DH key exchange protocol used in Signal, providing generalized frameworks that allow for post-quantum secure versions [11, 13, 22, 30], and analyzing its offline deniability guarantees [11, 22, 30, 48–50].

### 1.7 Summary of the Rest of the Paper

In Section 2 we provide our UC-based ideal functionalities in Figure 2. We put a lot of discussions around it for reader's convenience, and along the way explain why the transformations of Section 1.3 are insecure according to our definitions. In Section 3, we define the CKA primitive (capturing the public ratchet) and formally detail the two public ratchets used in the DR and TR (as described in Section 1.5), respectively, while only proving secure the latter.

Due to space limitations we defer the rest of the technical sections to the supplementary body. In Appendix A we provide the preliminaries containing mostly definitions borrowed from literature. In Appendix B we provide the other building blocks required for the DR (and TR), i.e., (i) we explain the (informal) properties required from the KDF used for the public ratchet (which we model as a random oracle to handle corruptions with messages in-transit; see Appendix B.1 for more discussion on this), (ii) we introduce FS-AEAD (formalizing the symmetric ratchet part), and (iii) we formally provide additional details on CKA, namely the security proof for the weaker public ratchet used in the DR. In Appendix C we detail the constructions, from the proper CKA and FS-AEAD notions, of protocols DR (Double Ratchet) and TR (Triple Ratchet), which use essentially the same presentation as ACD (fixing their error as described in $T_1(\mathsf{DR})$ and modelling the KDF as a random oracle). We also formally demonstrate the weakness of the DR with respect to our stronger functionality $\mathcal{F}_{\mathsf{TR}}$. In Appendix D we provide the full technical details of our transformations to the DR, their insecurity with respect to our functionality $\mathcal{F}_{\mathsf{DR}}$, and their security with respect to ACD's and/or CCD$^+$'s notion. In Appendix E we provide the security analyses of the Double Ratchet DR and Triple Ratchet TR, formalized in Theorem 5, which are essentially the same as that presented by ACD (but also using standard non-malleability arguments and programming the random oracle to handle corruptions with messages in-transit; again, see Appendix B.1).In Appendix F, we show how the techniques used in the TR can also be used to reduce the communication costs of practical UPKE schemes. Finally, Appendix G

contains technical descriptions of the UC framework, mostly borrowed from the literature, but adapted to our setting.

## 2 Defining Security of the Double Ratchet

In this section, we focus on obtaining an ideal functionality $\mathcal{F}_{\mathsf{DR}}$ that captures, as completely as possible, the security provided by the Double Ratchet algorithm. We emphasize that we study the security provided by the *strongest* implementation of the DR of which we are aware. For more on this, see Appendix D. We also provide an ideal functionality $\mathcal{F}_{\mathsf{TR}}$ that captures the security of our stronger TR protocol. Both functionalities are provided in Figure 2.

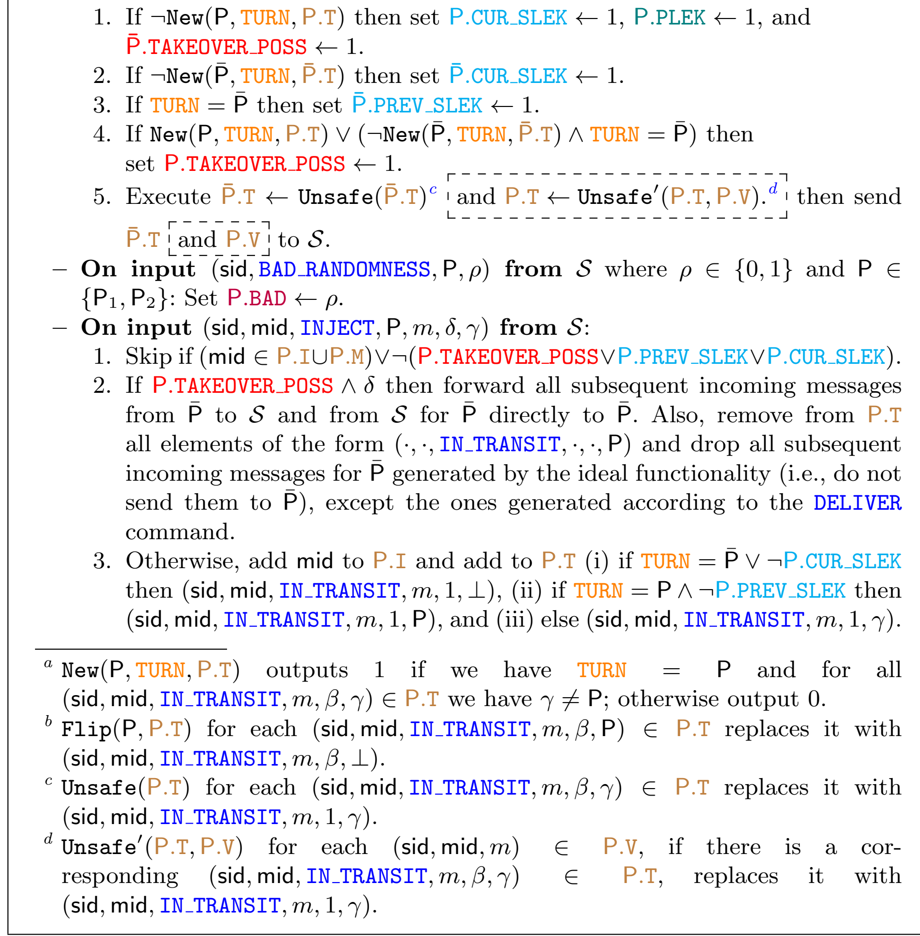
$\mathcal{F}_{\mathsf{DR}}$ and $\boxed{\mathcal{F}_{\mathsf{TR}}}$

**Notation:** The ideal functionality interacts with two parties $\mathsf{P}_1, \mathsf{P}_2$, and an ideal adversary $\mathcal{S}$. The ideal functionality initializes lists of *used message-ids* $\mathsf{P}_1.\mathtt{M}$, *in-transit* messages $\mathsf{P}_1.\mathtt{T}$, *adversarially injected* message-ids $\mathsf{P}_1.\mathtt{I}$, [and *vulnerable messages* $\mathsf{P}_1.\mathtt{V}$] sent by $\mathsf{P}_1$ to $\mathsf{P}_2$ to $\phi$. Analogously, lists $\mathsf{P}_2.\mathtt{M}$, $\mathsf{P}_2.\mathtt{T}$, $\mathsf{P}_2.\mathtt{I}$ [and $\mathsf{P}_2.\mathtt{V}$] are also initialized to $\phi$. The ideal functionality also initializes leakage flags of both $\mathsf{P}_1$ and $\mathsf{P}_2$ for their corresponding (i) public ratchet secrets: $\mathsf{P}_1.\mathtt{PLEK}$, $\mathsf{P}_2.\mathtt{PLEK}$, (ii) current sending epoch symmetric secrets: $\mathsf{P}_1.\mathtt{CUR_SLEK}$, $\mathsf{P}_2.\mathtt{CUR_SLEK}$, and (iii) previous sending epoch symmetric secrets: $\mathsf{P}_1.\mathtt{PREV_SLEK}$, $\mathsf{P}_2.\mathtt{PREV_SLEK}$, all to 0. Further, it initializes bad-randomness flags $\mathsf{P}_1.\mathtt{BAD}$, $\mathsf{P}_2.\mathtt{BAD}$ and takeover possible flags $\mathsf{P}_1.\mathtt{TAKEOVER_POSS}$, $\mathsf{P}_2.\mathtt{TAKEOVER_POSS}$ to 0. Finally, it initializes the turn flag $\mathtt{TURN}$ as $\perp$.

- **On input** $(\mathsf{sid}, \mathtt{SETUP})$ **from** $\mathsf{P}$ where $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$: Send $(\mathsf{sid}, \mathtt{SETUP}, \mathsf{P})$ to $\mathcal{S}$. When $\mathcal{S}$ returns $(\mathsf{sid}, \mathtt{SETUP})$ then set $\mathtt{TURN} \leftarrow \mathsf{P}$, and send $(\mathsf{sid}, \mathtt{INITIATED})$ to both $\mathsf{P}_1$ and $\mathsf{P}_2$. Ignore all future messages until this step is completed for $\mathsf{sid}$. Once this happens $\mathsf{P}$ can send the first message.
- **On input** $(\mathsf{sid}, \mathsf{mid}, \mathtt{SEND}, m)$ **from** $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$:
  1. Ignore if $\mathsf{mid} \in \mathsf{P}.\mathtt{M}$.
  2. [If $\bar{\mathsf{P}}.\mathtt{CUR_SLEK} \vee (\mathsf{P}.\mathtt{V} \neq \emptyset)$ then $\mathsf{P}.\mathtt{V} \cup \{(\mathsf{sid}, \mathsf{mid}, m)\}$]
  3. If $\mathtt{New}(\mathsf{P}, \mathtt{TURN}, \mathsf{P}.\mathtt{T})$[a] then set (i) $\mathsf{P}.\mathtt{PLEK} \leftarrow \mathsf{P}.\mathtt{BAD}$, (ii) $\mathsf{P}.\mathtt{CUR_SLEK} \leftarrow \bar{\mathsf{P}}.\mathtt{CUR_SLEK} \wedge (\mathsf{P}.\mathtt{PLEK} \vee \bar{\mathsf{P}}.\mathtt{PLEK})$, and (iii) $\bar{\mathsf{P}}.\mathtt{TAKEOVER_POSS} \leftarrow \mathsf{P}.\mathtt{CUR_SLEK}$.
  4. Add $\mathsf{mid}$ to $\mathsf{P}.\mathtt{M}$; if $\mathsf{mid} \notin \mathsf{P}.\mathtt{I}$ then add $(\mathsf{sid}, \mathsf{mid}, \mathtt{IN_TRANSIT}, m, \mathsf{P}.\mathtt{CUR_SLEK}, \mathtt{TURN})$ to $\mathsf{P}.\mathtt{T}$; and pass $(\mathsf{sid}, \mathsf{mid}, \mathtt{IN_TRANSIT}, \mathsf{P}, |m|, m')$ to $\mathcal{S}$ where $m' \leftarrow m$ if $\mathsf{P}.\mathtt{CUR_SLEK}$ and $\perp$ otherwise.
- **On input** $(\mathsf{sid}, \mathsf{mid}, \mathtt{DELIVER}, \mathsf{P}, m')$ **from** $\mathcal{S}$ where $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$:
  1. Find $(\mathsf{sid}, \mathsf{mid}, \mathtt{IN_TRANSIT}, m, \beta, \gamma) \in \mathsf{P}.\mathtt{T}$ and remove it from $\mathsf{P}.\mathtt{T}$. Skip rest of the steps if no such entry is found.
  2. If $\gamma = \mathsf{P}$ then set (i) $\mathtt{TURN} \leftarrow \bar{\mathsf{P}}$, (ii) $\mathsf{P}.\mathtt{T} \leftarrow \mathtt{Flip}(\mathsf{P}, \mathsf{P}.\mathtt{T})$,[b] (iii) $\mathsf{P}.\mathtt{PREV_SLEK} \leftarrow 0$, (iv) $\bar{\mathsf{P}}.\mathtt{PREV_SLEK} \leftarrow \bar{\mathsf{P}}.\mathtt{CUR_SLEK}$, (v) $\bar{\mathsf{P}}.\mathtt{CUR_SLEK} \leftarrow 0$, (vi) $\bar{\mathsf{P}}.\mathtt{PLEK} \leftarrow 0$, (vii) $\mathsf{P}.\mathtt{TAKEOVER_POSS} \leftarrow 0$, [and (viii) $\bar{\mathsf{P}}.\mathtt{V} \leftarrow \emptyset$.]
  3. If $\beta = 1$ then set $m \leftarrow m'$. Send $(\mathsf{sid}, \mathsf{mid}, \mathtt{DELIVER}, m)$ to $\bar{\mathsf{P}}$.

---

- **On input** $(\mathsf{sid}, \mathtt{LEAK}, \mathsf{P})$ **from** $\mathcal{S}$ where $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$:



1. If $\neg\texttt{New}(\mathsf{P}, \texttt{TURN}, \mathsf{P}.\texttt{T})$ then set $\mathsf{P}.\texttt{CUR_SLEK} \leftarrow 1$, $\mathsf{P}.\texttt{PLEK} \leftarrow 1$, and $\bar{\mathsf{P}}.\texttt{TAKEOVER_POSS} \leftarrow 1$.
2. If $\neg\texttt{New}(\bar{\mathsf{P}}, \texttt{TURN}, \bar{\mathsf{P}}.\texttt{T})$ then set $\bar{\mathsf{P}}.\texttt{CUR_SLEK} \leftarrow 1$.
3. If $\texttt{TURN} = \bar{\mathsf{P}}$ then set $\bar{\mathsf{P}}.\texttt{PREV_SLEK} \leftarrow 1$.
4. If $\texttt{New}(\mathsf{P}, \texttt{TURN}, \mathsf{P}.\texttt{T}) \vee (\neg\texttt{New}(\bar{\mathsf{P}}, \texttt{TURN}, \bar{\mathsf{P}}.\texttt{T}) \wedge \texttt{TURN} = \bar{\mathsf{P}})$ then set $\mathsf{P}.\texttt{TAKEOVER_POSS} \leftarrow 1$.
5. Execute $\bar{\mathsf{P}}.\texttt{T} \leftarrow \texttt{Unsafe}(\bar{\mathsf{P}}.\texttt{T})$[c] and $\mathsf{P}.\texttt{T} \leftarrow \texttt{Unsafe}'(\mathsf{P}.\texttt{T}, \mathsf{P}.\texttt{V})$.[d] then send $\bar{\mathsf{P}}.\texttt{T}$ and $\mathsf{P}.\texttt{V}$ to $\mathcal{S}$.

– **On input** $(\mathsf{sid}, \texttt{BAD_RANDOMNESS}, \mathsf{P}, \rho)$ **from** $\mathcal{S}$ where $\rho \in \{0,1\}$ and $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$: Set $\mathsf{P}.\texttt{BAD} \leftarrow \rho$.

– **On input** $(\mathsf{sid}, \mathsf{mid}, \texttt{INJECT}, \mathsf{P}, m, \delta, \gamma)$ **from** $\mathcal{S}$:

1. Skip if $(\mathsf{mid} \in \mathsf{P}.\texttt{I} \cup \mathsf{P}.\texttt{M}) \vee \neg(\mathsf{P}.\texttt{TAKEOVER_POSS} \vee \mathsf{P}.\texttt{PREV_SLEK} \vee \mathsf{P}.\texttt{CUR_SLEK})$.
2. If $\mathsf{P}.\texttt{TAKEOVER_POSS} \wedge \delta$ then forward all subsequent incoming messages from $\bar{\mathsf{P}}$ to $\mathcal{S}$ and from $\mathcal{S}$ for $\bar{\mathsf{P}}$ directly to $\bar{\mathsf{P}}$. Also, remove from $\mathsf{P}.\texttt{T}$ all elements of the form $(\cdot, \cdot, \texttt{IN_TRANSIT}, \cdot, \cdot, \mathsf{P})$ and drop all subsequent incoming messages for $\bar{\mathsf{P}}$ generated by the ideal functionality (i.e., do not send them to $\bar{\mathsf{P}}$), except the ones generated according to the $\texttt{DELIVER}$ command.
3. Otherwise, add $\mathsf{mid}$ to $\mathsf{P}.\texttt{I}$ and add to $\mathsf{P}.\texttt{T}$ (i) if $\texttt{TURN} = \bar{\mathsf{P}} \vee \neg\mathsf{P}.\texttt{CUR_SLEK}$ then $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, 1, \bot)$, (ii) if $\texttt{TURN} = \mathsf{P} \wedge \neg\mathsf{P}.\texttt{PREV_SLEK}$ then $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, 1, \mathsf{P})$, and (iii) else $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, 1, \gamma)$.

[a] $\texttt{New}(\mathsf{P}, \texttt{TURN}, \mathsf{P}.\texttt{T})$ outputs 1 if we have $\texttt{TURN} = \mathsf{P}$ and for all $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, \beta, \gamma) \in \mathsf{P}.\texttt{T}$ we have $\gamma \neq \mathsf{P}$; otherwise output 0.

[b] $\texttt{Flip}(\mathsf{P}, \mathsf{P}.\texttt{T})$ for each $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, \beta, \mathsf{P}) \in \mathsf{P}.\texttt{T}$ replaces it with $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, \beta, \bot)$.

[c] $\texttt{Unsafe}(\mathsf{P}.\texttt{T})$ for each $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, \beta, \gamma) \in \mathsf{P}.\texttt{T}$ replaces it with $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, 1, \gamma)$.

[d] $\texttt{Unsafe}'(\mathsf{P}.\texttt{T}, \mathsf{P}.\texttt{V})$ for each $(\mathsf{sid}, \mathsf{mid}, m) \in \mathsf{P}.\texttt{V}$, if there is a corresponding $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, \beta, \gamma) \in \mathsf{P}.\texttt{T}$, replaces it with $(\mathsf{sid}, \mathsf{mid}, \texttt{IN_TRANSIT}, m, 1, \gamma)$.

Fig. 2: The ideal functionalities $\mathcal{F}_{\mathsf{DR}}$ and $\mathcal{F}_{\mathsf{TR}}$, respectively.

### 2.1 Honest Execution

We start with a simplified view of the functionality where only the first three commands, namely SETUP, SEND, and DELIVER are executed. In other words, we consider a restricted view of the ideal functionality where leakage, bad randomness and injection attacks are not allowed. The adversary is still allowed to delay, reorder, and drop messages at will.

*SETUP Command.* This command can be initiated by either $\mathsf{P} = \mathsf{P}_1$ or $\mathsf{P} = \mathsf{P}_2$, and allows for initializing the communication channel between $\mathsf{P}$ and $\bar{\mathsf{P}}$. Looking ahead, in the real-world protocol, this initialization will involve sharing cryptographic secrets between the real-world $\mathsf{P}$ and real-world $\bar{\mathsf{P}}$, then properly initializing their states using these secrets. While the actual Signal protocol

uses the X3DH key exchange [41] for this, the focus of our work is to analyze the security and functionality of the double ratchet algorithm, and not X3DH. Therefore, we present a simple description for the SETUP command, that may be stronger than what X3DH achieves, but nonetheless suffices for our purposes.

We note that both $\mathsf{P}_1$ and $\mathsf{P}_2$ must receive $(\mathsf{sid}, \texttt{INITIATED})$ before the communication between them can proceed. Turn status flag TURN is set to the initiator $\mathsf{P}$ to denote that $\mathsf{P}$ will be the first party to send a message.

*SEND Command.* This command allows $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$ to send a message $m$, under a unique assigned message id $\mathsf{mid}$, to $\bar{\mathsf{P}}$. Naturally, the ideal functionality only allows $\mathsf{P}$ to send one message under each such $\mathsf{mid}$, which it ensures by aborting in Step 1 if $\mathsf{mid}$ is already in list $\texttt{P.M}$, and subsequently adding $\mathsf{mid}$ to $\texttt{P.M}$ in Step 4 otherwise. Now, this message might be dropped or delayed while in transit. Thus, at this point, the message is only added to the in-transit list $\texttt{P.T}$ (Step 4) and the ideal-functionality waits for the instruction from the ideal-world adversary on when this message is to be delivered (if at all).

Observe that the last element of each tuple in $\texttt{P.T}$ is TURN: the turn status when $\mathsf{P}$ attempted to send this message (i.e., when it was added to $\texttt{P.T}$). Looking ahead, this element is used in helper function $\texttt{New}(\mathsf{P}, \texttt{TURN}, \texttt{P.T})$ within SEND (Step 3) to determine whether $\mathsf{P}$ is initiating a new epoch when sending a message and, if so, the (in)security of the new epoch. When discussing the DELIVER command below, we will explain the role the last element of $\texttt{P.T}$ plays in the logic of $\texttt{New}(\mathsf{P}, \texttt{TURN}, \texttt{P.T})$ and further understand its role elsewhere in the functionality.

Finally, as typical with encryption, in the real-world the length of the encrypted message is often leaked by the ciphertext. Thus, the ideal functionality leaks the length of sent messages to the ideal adversary.

*DELIVER Command.* This command allows the ideal adversary to instruct the ideal functionality that a certain message, with unique message id $\mathsf{mid}$, is no longer in-transit, and must be delivered to the recipient *immediately*, whether or not previously sent messages have already been delivered (thus transformation $T_6(\mathsf{DR})$ cannot realize the ideal functionality). The ideal functionality restricts the ideal adversary to delivering the message associated with this $\mathsf{mid}$ only once, which reflects the forward security of the $\mathsf{DR}$ – once a message is delivered, the recipient should no longer be able to decrypt it (in case she is leaked on afterwards). This is done by removing the entry for $\mathsf{mid}$ from $\texttt{P.T}$ when it is delivered, so that subsequent deliveries cannot occur (Step 1).

As part of the delivery process (Step 2), the ideal functionality also checks if TURN was set to $\mathsf{P}$ when this message was sent. If so, the message was indeed the first of $\mathsf{P}$'s newest epoch that is delivered to $\bar{\mathsf{P}}$ (out of possibly many messages that can be the first delivered in the epoch). Thus, subsequently, it will next be $\bar{\mathsf{P}}$'s turn to start a new epoch. So, if this is the case, then TURN is flipped to $\bar{\mathsf{P}}$. Additionally, helper function $\texttt{Flip}(\mathsf{P}, \texttt{P.T})$ flips the last entry of each message from $\mathsf{P}$ to $\bar{\mathsf{P}}$ in $\texttt{P.T}$ to $\perp$. This is done so that subsequently, when $\mathsf{P}$ starts its next sending epoch, $\texttt{New}(\mathsf{P}, \texttt{TURN}, \texttt{P.T})$ will return 1: TURN will flip back to $\mathsf{P}$ once

a message of $\bar{\mathsf{P}}$'s next sending epoch is delivered to $\mathsf{P}$ for the first time, and there will be no element in $\mathtt{P.T}$ whose last entry is $\mathsf{P}$. (Note: before $\mathsf{P}$ receives a message for $\bar{\mathsf{P}}$'s next sending epoch, $\mathsf{P}$'s sent messages will not start a new epoch, as captured by $\mathtt{New}(\mathsf{P}, \mathtt{TURN}, \mathtt{P.T})$, since $\mathtt{TURN}$ will be set to $\bar{\mathsf{P}}$.)

We also note that since UC modelling typically allows the adversary to control the communication network [14] (and thus decide when ciphertexts should be delivered), there are some *useless* protocols that may realize $\mathcal{F}_{\mathsf{DR}}$ and $\mathcal{F}_{\mathsf{TR}}$. We define *useless* protocols as those in which with any PPT environment and adversary, parties do not generate output (i.e., not even a special reject symbol, like $\perp$, representing failed authentication) for at least one ciphertext delivery, with non-negligible probability. However, we can trivially rule out such useless protocols, so that all protocols that do realize $\mathcal{F}_{\mathsf{DR}}$ or $\mathcal{F}_{\mathsf{TR}}$ and that are *not* useless indeed generate the correct output immediately upon every delivery of a ciphertext from the adversary, with all-but-negligible probability.

### 2.2 Execution with an Unrestricted Adversary

In addition to delaying, reordering, and dropping messages, we assume that the real-world adversary can: (i) provide bad randomness for both parties, (ii) leak their secret states; possibly multiple times at various points in the execution, (iii) tamper with in-transit messages between the parties, and (iv) attempt to inject messages on behalf of both parties. Here, we explain how the ideal functionality captures this behavior.

*The Ideal Functionality's Flags.* The ideal functionality uses several binary flags to properly capture adversarial behavior. The functionality initializes all of them to 0. Binary flag $\mathtt{P.BAD}$ captures bad randomness for party $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$. Naturally, $\mathtt{P.BAD}$ is set to 0 or 1 when the ideal-world adversary issues a $(\mathsf{sid}, \mathit{cmdBAD_RANDOMNESS}, \mathsf{P}, \rho)$ command to the ideal functionality, depending on the the value of $\rho$. If $\mathtt{P.BAD}$ is set to 1 then $\mathsf{P}$ is provided with bad randomness (by the adversary, c.f. Appendix G) when she tries to sample some (thus rendering transformation $T_5(\mathsf{DR})$ insecure). Otherwise, $\mathsf{P}$ samples fresh randomness.

The ideal functionality further utilizes the following binary flags for each party $\mathsf{P} \in \{\mathsf{P}_1, \mathsf{P}_2\}$ to capture the rest of the possible adversarial behavior. We first introduce their real-world semantic meaning here before explaining (i) their evolution within the ideal functionality as a result of the ideal world adversary's behavior, then (ii) how they thus allow the ideal functionality to determine security for the session.

- $\mathtt{P.PLEK}$ (Public Ratchet Secrets Leakage): If $\mathtt{P.PLEK}$ is set to 1 then $\mathsf{P}$'s public ratchet secrets are leaked to the real-world adversary. Otherwise, they should be hidden from the real-world adversary.
- $\mathtt{P.CUR_SLEK}$ (Current Sending Symmetric Ratchet Secrets Leakage): If $\mathtt{P.CUR_SLEK}$ is set to 1 then the symmetric ratchet secrets of $\mathsf{P}$'s *current* sending epoch are leaked to the real-world adversary. Otherwise, they should be hidden from the real-world adversary

- P.PREV_SLEK (Previous Sending Symmetric Ratchet Secrets Leakage): If P.PREV_SLEK is set to 1 then the symmetric ratchet secrets of the *previous* sending epoch of P are leaked to the real-world adversary. Otherwise, they should be hidden from the real-world adversary.
- P.TAKEOVER_POSS (Takeover Possible): If P.TAKEOVER_POSS is set to 1 then the real-world adversary has the option to take over the role of P in the conversation with $\bar{P}$. Otherwise, the real-world adversary should not have this option.

*How the Flags are Affected by Leakages.* We first describe how a leakage on one of the parties $P \in \{P_1, P_2\}$ affects the above flags. For P.PLEK, when $\texttt{New}(P, \texttt{TURN}, P.\texttt{T}) = 1$, it is P's turn to start her next sending epoch, but she has not yet started it. Thus she does not currently have any public ratchet secret state (just $\bar{P}$'s public value), so there is no effect on P.PLEK if leakage on P occurs in this case. If $\texttt{New}(P, \texttt{TURN}, P.\texttt{T}) = 0$ when leakage on P occurs, P of course does have a public ratchet secret state, as she needs to be able to receive a message for $\bar{P}$'s next sending turn; thus in command LEAK, the ideal functionality sets P.PLEK to 1 (Step 1). Since P never stores $\bar{P}$'s public ratchet secrets, there is never any effect on $\bar{P}$.PLEK when P's state is leaked.

For P.CUR_SLEK, the functionality has similar behavior. If $\texttt{New}(P, \texttt{TURN}, P.\texttt{T}) = 1$ when leakage on P occurs, P has not yet generated the secrets for her next sending epoch, so P.CUR_SLEK is not modified. Otherwise, P has started the epoch, and so she stores the corresponding secrets in order to send new messages for the epoch; thus in command LEAK, we set P.CUR_SLEK to 1 (Step 1). Additionally, if $\texttt{New}(\bar{P}, \texttt{TURN}, \bar{P}.\texttt{T}) = 1$ then $\bar{P}$ has not yet generated the secrets for her next sending epoch, so $\bar{P}$.CUR_SLEK is not modified. Otherwise, $\bar{P}$ has indeed started the epoch, in which case P must be able to derive the epoch's symmetric secrets (possibly using in-transit messages, which the adversary has), and thus in command LEAK we set $\bar{P}$.CUR_SLEK to 1 (Step 2).

For P.PREV_SLEK, since in the most secure version of the DR, P only ever stores the secrets for her current sending epoch (if she has indeed started it), leakage on P has no effect on P.PREV_SLEK. However, once it is $\bar{P}$'s turn to start a new sending epoch, P still stores the secrets of $\bar{P}$'s previous sending epoch (in case she needs to receive messages for it; she does not yet know $\bar{P}$ will never again send a message for that epoch), until she receives a message in $\bar{P}$'s new epoch for the first time. Therefore, if $\texttt{TURN} = \bar{P}$ then in command LEAK, we set $\bar{P}$.PREV_SLEK to 1 (Step 3); otherwise, if $\texttt{TURN} = P$, $\bar{P}$.PREV_SLEK is not modified.

Finally, for P.TAKEOVER_POSS, if it is P's turn to start a new sending epoch, then of course a leakage on P will enable the adversary to forge the first message of this new epoch and thus influence the subsequent state of $\bar{P}$ upon delivery such that the adversary can take over P's role in the conversation (if it wishes). This is because the adversary will obtain the double ratchet root key, and can thus send such a message herself. Also, note that this key is derived from (i) P's previous state before she received any message for $\bar{P}$'s newest epoch and (ii) any message of $\bar{P}$'s newest sending epoch. Thus, additionally, if P is leaked while any message from $\bar{P}$'s newest epoch is in-transit, but before P receives any

such message, then the adversary can obtain the root key as above, and so will have the ability to forge the first message of P's next sending epoch. Therefore, in command LEAK, if $\mathtt{New}(\mathsf{P}, \mathtt{TURN}, \mathsf{P}.\mathtt{T}) = 1$, or $\mathtt{New}(\bar{\mathsf{P}}, \mathtt{TURN}, \bar{\mathsf{P}}.\mathtt{T}) = 0$ and $\mathtt{TURN} = \bar{\mathsf{P}}$, then we set P.TAKEOVER_POSS to 1 (Step 4). Otherwise, if P has already sent the first message of the epoch, and $\bar{\mathsf{P}}$ has not yet started her next sending epoch, leakage on P does not reveal the root key, so P.TAKEOVER_POSS is not modified. In the former case, this is because P deletes the key after sending the message, and in the latter case, this is because the key does not yet exist. Furthermore, if P has indeed sent a message for her current sending epoch, then a leakage on P will provide the adversary with the new root key. The adversary will therefore be able to forge the first message for $\bar{\mathsf{P}}$'s next sending epoch. So, if $\mathtt{New}(\mathsf{P}, \mathtt{TURN}, \mathsf{P}.\mathtt{T}) = 0$ then in command LEAK, we additionally set $\bar{\mathsf{P}}$.TAKEOVER_POSS to 1 (Step 1). Otherwise, if it is P's turn to start a new epoch, and she has not yet started it, then the new root key has not yet been generated, so $\bar{\mathsf{P}}$.TAKEOVER_POSS is not modified.

*How the Flags are Affected by Epoch Initialization.* The effects on the ideal functionality's flags of epoch initialization via a SEND command are determined in Step 3 of the command. First, if $\mathsf{P}.\mathtt{BAD} = 1$ when starting a new epoch (i.e. P uses bad randomness to start it), then we of course set P.PLEK to 1 (In the TR we may still here have security of P's public ratchet secret state, but we choose to capture slightly weaker security for simplicity); otherwise we set P.PLEK to 0. Now, consider the privacy of the root key when $\bar{\mathsf{P}}$.CUR_SLEK is 1 and P is initializing a new epoch:

- If $\bar{\mathsf{P}}$.CUR_SLEK was set to 1 when $\bar{\mathsf{P}}$ initialized her newest epoch (as we explain below), then the root key must have been leaked in addition to the corresponding symmetric ratchet secrets, since they are both part of the same KDF output.
- If $\bar{\mathsf{P}}$.CUR_SLEK was set to 1 as a result of a leakage on P, then the root key must have been also leaked, since P needs it to start her new sending epoch.
- Finally, if $\bar{\mathsf{P}}$.CUR_SLEK was set to 1 as a result of a leakage on $\bar{\mathsf{P}}$, then the root key must have been also leaked, since $\bar{\mathsf{P}}$ needs it to receive a message for P's new sending epoch.

So, if $\bar{\mathsf{P}}$.CUR_SLEK is 1 when P initializes her new sending epoch, then it must be that the root key is leaked. Thus, only if P and $\bar{\mathsf{P}}$ have a secure key exchange can security for the DR be recovered, which only happens if both P.PLEK and $\bar{\mathsf{P}}$.PLEK are 0, i.e., their public ratchet secrets are both hidden from the adversary. In this case, we set P.CUR_SLEK to 0; otherwise, we set it to 1. If $\bar{\mathsf{P}}$.CUR_SLEK is 0 at the time of initialization, then the root key must be hidden. This is because if not, then the current symmetric ratchet secrets of $\bar{\mathsf{P}}$ would also not be hidden, since they were part of the same KDF output when $\bar{\mathsf{P}}$ started her latest sending epoch, and there were no subsequent leakages on either party. So we set P.CUR_SLEK to 0 upon initialization, in this case.

Finally, if we do indeed set P.CUR_SLEK to 1 at this time, as we noted above, this means that the new root key is known by the adversary, and thus

the adversary could forge the first message for $\bar{\mathsf{P}}$'s next turn; otherwise the root key is hidden, and so the adversary does not have this ability. So, we set $\bar{\mathsf{P}}$.TAKEOVER_POSS $\leftarrow$ P.CUR_SLEK.

*How the Flags are Affected by Epoch Termination.* When the ideal adversary issues a DELIVER command for the first message of P's newest sending epoch, the ideal functionality needs to properly evolve the flags it uses to capture adversarial behavior (Step 2). First, when such a delivery occurs, $\bar{\mathsf{P}}$'s latest sending epoch terminates, as her next message will be sent in a new epoch. To reflect this, upon such a delivery, the ideal functionality sets $\bar{\mathsf{P}}$.PREV_SLEK $\leftarrow$ $\bar{\mathsf{P}}$.CUR_SLEK. Also, since $\bar{\mathsf{P}}$ deletes her public ratchet secrets upon reception of such a message, and her newest epoch has not actually started at this point, the functionality sets $\bar{\mathsf{P}}$.CUR_SLEK $\leftarrow 0$ and $\bar{\mathsf{P}}$.PLEK $\leftarrow 0$.

Furthermore, in the DR, P includes in each message of an epoch the number of messages she sent in her previous epoch (see Appendix C). Thus, once $\bar{\mathsf{P}}$ receives such a message in the DR, she knows exactly how many messages P sent in her previous epoch. So, the adversary can no longer inject messages in P's previous epoch (just modify them) and there is no more adversarial action possible for that epoch, so the functionality sets P.PREV_SLEK $\leftarrow 0$. Finally, since a message for P's newest sending epoch has indeed been delivered at this point, the secrets needed to start her next sending epoch are yet to be determined. Thus, the adversary cannot yet forge a message to start her next sending epoch, so the functionality sets P.TAKEOVER_POSS $\leftarrow 0$.

*Determining New Messages' Privacy and Authenticity.* We know from above that if P.CUR_SLEK $= 1$, then P's current symmetric ratchet secrets are leaked to the adversary. Thus, if P issues a SEND command for message $m$ with id mid, and P.CUR_SLEK $= 1$, then the ideal functionality leaks the corresponding message to the ideal adversary (Step 4). Additionally, the ideal functionality sets the penultimate element of mid's entry in P.T to 1. This will allow the ideal adversary to modify the message associated with mid upon delivery: the adversary will issue a DELIVER command for mid to the functionality with input modified message $m'$, which will then be delivered $\bar{\mathsf{P}}$, instead of $m$ (Step 3).

Otherwise, if P.CUR_SLEK $= 0$ when P issues the SEND command, then the ideal functionality only leaks the message length to the adversary and sets the penultimate element of the corresponding entry of P.T to 0, ensuring (for now) privacy and authenticity of $m$.

*The Consequences of Leakages.* When the adversary leaks on P in the real-world, the privacy of in-transit messages from $\bar{\mathsf{P}}$ to P is no longer guaranteed, since P must preserve all keys that will be necessary for authenticating and decrypting them. Therefore, when the ideal adversary issues a LEAK command on P, the ideal functionality leaks the in-transit messages from $\bar{\mathsf{P}}$ to P, P.T, to the ideal adversary, and allows the ideal adversary to modify them in the future (Step 5). The ideal functionality accomplishes the latter using helper function Unsafe($\bar{\mathsf{P}}$.T) which sets the penultimate element of each in-transit message of $\bar{\mathsf{P}}$.T to 1. As a

result, the ideal adversary can modify these in-transit messages in the DELIVER command, as described above. Note that only *in-transit* messages from $\bar{\mathsf{P}}$ to P are affected (thus rendering transformation $T_2(\mathsf{DR})$ insecure).

*Vulnerable Messages in the* DR. As explained in Section 1.4, if in the DR, the root key is leaked when it is P's turn to start a new sending epoch, but P has not yet started it, then the messages of that epoch are *vulnerable*. This means that if P is leaked on before P receives a message of $\bar{\mathsf{P}}$'s next sending epoch for the first time, the messages that P sent in her own epoch become insecure.

To capture this, the ideal functionality in the SEND command adds messages to list P.V if they are indeed vulnerable (Step 2). At the start of the epoch, this is the case if $\bar{\mathsf{P}}$.CUR_SLEK $= 1$ (as explained above); in the middle of the epoch, this is the case if P.V is non-empty. Hence, if the adversary issues a LEAK command on P, in addition to the consequences of the above paragraph, the ideal functionality *also* leaks P.V and allows for future modification of its elements that are still in-transit (Step 5). The latter is accomplished via helper function Unsafe′(P.T, P.V), similarly as in Unsafe($\bar{\mathsf{P}}$.T). Note that this scenario, and the one above, are the only ones in which secure, in-transit messages are leaked to the adversary and/or subject to modification (thus transformation $T_4(\mathsf{DR})$ is insecure). Finally, if the adversary issues a DELIVER command for the first message of $\bar{\mathsf{P}}$'s next sending epoch, the ideal functionality sets P.V $= \emptyset$: P properly deletes the secrets which make those messages vulnerable at this time.

*Injections and Takeovers.* If P.CUR_SLEK $= 1$ or P.PREV_SLEK $= 1$, then the adversary has the secrets required to inject its own messages into P's current or previous sending epoch, respectively. Also, if P.TAKEOVER_POSS $= 1$, then the adversary can forge the first message to be delivered in P's next sending epoch to $\bar{\mathsf{P}}$. In either case, the ideal adversary issues the INJECT command to inject message $m$ under unique message id mid on behalf of P. Of course, the ideal functionality only allows the adversary to inject one message under each such mid, which it ensures by aborting in Step 1 if mid is already in P.I, and adding it to P.I in Step 3 if not. The ideal functionality also aborts if a message with message id mid was already sent by P, i.e., it is in P.M, in which case injection of mid is not allowed, only modification. If the ideal adversary injects a message with id mid that is not a takeover forgery, then before actual delivery of the injection occurs, a corresponding entry is added to P.T.

Now, if P.TAKEOVER_POSS $= 1$, and the ideal adversary inputs $\delta = 1$ to the INJECT command, indicating that it wishes to takeover for P, then the ideal functionality thereafter directly forwards messages sent from P to the ideal adversary, and vice versa (Step 2).

If the ideal adversary injects a message with id mid that is not a takeover forgery, then before actual delivery of the injection occurs, a corresponding entry is added to P.T. However, the ideal functionality has to be careful to set the last element of this entry correctly:

- If TURN $= \bar{\mathsf{P}}$, then the first message of P's current sending epoch has already been delivered to $\bar{\mathsf{P}}$. Thus, the last element of the entry is set to $\perp$, so that

if TURN is flipped to P, the entry's subsequent delivery does not prematurely flip TURN back. Moreover, if P.CUR_SLEK = 0, then the adversary must be injecting into P's previous sending epoch, so for the same reason as above, we set its last entry to $\perp$.
- If P.PREV_SLEK = 0 and TURN = P, then the adversary must be injecting into P's current sending epoch, and moreover, it might be that the injected message could be the first of the epoch delivered to $\bar{\mathsf{P}}$. Therefore, we set TURN to P.
- If neither of the above are true, i.e., TURN = P, P.PREV_SLEK = 1, and P.CUR_SLEK = 1, then it could be that the adversary is injecting into *either* P's previous or current sending epoch. Therefore, the ideal adversary specifies its choice of the last element with the last input $\gamma$ to the INJECT command.

Actual delivery of injections is then handled in the DELIVER command, in the same simple manner as specified in the Honest Execution Section (Section 2.1). Namely, delivery of injected message with message id mid is done by removing it from P.T (if such an entry exists), and sending it to $\bar{\mathsf{P}}$. The functionality works this way in order to capture the scenario in which the real-world adversary modifies the first message of a new sending epoch for P to inform $\bar{\mathsf{P}}$ that P's last sending epoch contains more messages than it actually does. Therefore, the real-world adversary will be able to in the future inject such additional messages whenever it wants. The ideal-world adversary thus issues an INJECT command for all of these message ids at the time of the first modification, so that later it can actually send them to $\bar{\mathsf{P}}$ using DELIVER commands (regardless of the status of the functionality's flags at that time).

If an injected message with id mid is indeed added to P.T, then the ideal functionality needs to also make sure that P can send a message with the same mid (since it does not know about the injection), but not allow the ideal adversary to deliver two messages with the same mid (since $\bar{\mathsf{P}}$ will only accept one such message in the DR). Therefore, in the SEND command, the ideal functionality checks if mid $\notin$ P.I and if so adds the corresponding message to P.T as in the honest execution. However, if mid is in P.I, the ideal functionality does not add the corresponding message to P.T, but still sends the length of the message (and the message itself if P.CUR_SLEK = 1) to the ideal adversary, mirroring that a ciphertext is still created in the real-world.

## 3 Continuous Key Agreement

In this section we formalize the main constructive contribution of our paper: the stronger, but virtually as efficient, public ratchet (and its mini ratchet) used by the Triple Ratchet protocol. More specifically, we first define (a version of) the *continuous key agreement* (CKA) primitive, introduced by ACD, which provides secrets for updates of the public ratchet. We provide two notions of security, that which is used by the DR and that which is used by the (stronger) TR. We also compare our definition to that of ACD. Then, we provide our CKA construction $\mathsf{CKA}^+$ used in the TR and show it is secure according to the stronger definition,

via the *strong-DH* (StDH) assumption [1] in the random oracle model.[1] The StDH assumption is: given random and independent group elements $g^a, g^b$, and access to oracle $\mathbf{ddh}(g^a, \cdot, \cdot)$, which on input $X, Y$ returns 1 if $X^a = Y$ and 0 otherwise, it is hard to compute $g^{ab}$.

**Defining CKA** At a high level, CKA is a synchronous two-party protocol between $\mathsf{P}_1$ and $\mathsf{P}_2$. Odd rounds $i$ consist of $\mathsf{P}_1$ sending and $\mathsf{P}_2$ receiving a message $T_i$, whereas in even rounds, $\mathsf{P}_2$ is the sender and $\mathsf{P}_1$ the receiver. Each round $i$ also produces a key $I_i$, which is output by the sender upon sending $T_i$ and by the receiver upon receiving $T_i$.

**Definition 1.** *A* continuous-key-agreement (CKA) scheme *is a quadruple of algorithms* $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}P}_1, \mathsf{CKA\text{-}Init\text{-}P}_2, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$*, where*

- $\mathsf{CKA\text{-}Init\text{-}P}_1$ *(and similarly* $\mathsf{CKA\text{-}Init\text{-}P}_2$*) takes a key $k$ and produces an initial state* $\gamma^{\mathsf{P}_1} \leftarrow \mathsf{CKA\text{-}Init\text{-}P}_1(k)$ *(and* $\gamma^{\mathsf{P}_2}$*),*
- $\mathsf{CKA\text{-}S}$ *takes a state $\gamma$, and produces a new state, message, and key* $(\gamma', T, I) \stackrel{\$}{\leftarrow} \mathsf{CKA\text{-}S}(\gamma)$*, and*
- $\mathsf{CKA\text{-}R}$ *takes a state $\gamma$ and message $T$ and produces new state and a key* $(\gamma', I) \leftarrow \mathsf{CKA\text{-}R}(\gamma, T)$*.*

*Denote by $\mathcal{K}$ the space of initialization keys $k$ and by $\mathcal{I}$ the space of CKA keys $I$.*

*Correctness.* A CKA scheme is correct if in the security game in Figure 3 (explained below), $\mathsf{P}_1$ and $\mathsf{P}_2$ always, i.e., with probability 1, output the same key in every round.

*Security.* The security property we will require a CKA scheme to satisfy is that conditioned on the transcript $T_1, T_2, \ldots$, the keys $I_1, I_2, \ldots$ are unrecoverable. An attacker against a CKA scheme is required to be passive, i.e., may not modify the messages $T_i$. However, it is given the power to possibly (1) control the random coins used by the sender and (2) leak the current state of either party. Given the capabilities of the adversary, it is easy to see that some keys $I_i$ would be recoverable. The security guarantee offered by the CKA scheme would then be that even given the transcript $T_1, T_2, \ldots$, assuming certain "fine-grained" conditions around when the adversary controls the randomness used by parties and when the adversary learns the state of parties, most keys $I_1, I_2, \ldots$ are unrecoverable. It will also be the case that parties thus recover from such bad randomness and leakage issued by the adversary.

The formal security game for CKA is provided in Figure 3. It begins with a call to the **init** oracle, which initializes the states of both parties, and defines

[1] We do not provide CKA schemes secure according to our definitions based on LWE or generic KEMs, as in ACD. However, we note that our stronger scheme $\mathsf{CKA}^+$ is intuitively at least as strong as their construction from generic KEMs, but more efficient.

**Security Games for CKA**

```
init (t*)
  k <-$ K
  γ_0^P1 <- CKA-Init-P1(k)
  γ_0^P2 <- CKA-Init-P2(k)
  t_P1, t_P2 <- 0
  Recv-State[*] <- ⊥

corr-P1
  req allow-corr_P1
  return γ_{t_P1}^P1

send-P1
  t_P1 ++
  (γ_{t_P1}^P1, T, I) <-$ CKA-S(γ_{t_P1}^P1)
  Recv-State[t_P1 + 1] <- γ_{t_P1}^P1
  return (T, I)

send-P1'(r)
  t_P1 ++
  req allow-bad-rand_P1
  (γ_{t_P1}^P1, T, I) <- CKA-S(γ_{t_P1 -1}^P1; r)
  Recv-State[t_P1 + 1] <- γ_{t_P1}^P1
  return (T, I)

receive-P1
  t_P1 ++
  (γ_{t_P1}^P1, *) <- CKA-R(γ_{t_P1 -1}^P1, T)

chall-P1
  t_P1 ++
  req t_P1 = t*
  (γ_{t_P1}^P1, T, I) <-$ CKA-S(γ_{t_P1 -1}^P1)
  return T

test (t, T, I)
  req Recv-State[t] ≠ ⊥
  if (*, I) <- CKA-R(Recv-State[t], T)
    return 1
  else
    return 0
```

$\mathsf{allow\text{-}corr}_{\mathsf{P}_1}, \mathsf{allow\text{-}bad\text{-}rand}_{\mathsf{P}_1} :\Longleftrightarrow \begin{cases} t_{\mathsf{P}_1} \neq t^* & t^* \text{ is odd} \\ t_{\mathsf{P}_1} \neq t^* - 1 & t^* \text{ is even} \end{cases}$

$\mathsf{allow\text{-}corr}_{\mathsf{P}_2}, \mathsf{allow\text{-}bad\text{-}rand}_{\mathsf{P}_2} :\Longleftrightarrow \begin{cases} t_{\mathsf{P}_2} \neq t^* - 1 & t^* \text{ is odd} \\ t_{\mathsf{P}_2} \neq t^* & t^* \text{ is even} \end{cases}$

$\mathsf{allow\text{-}corr}_{\mathsf{P}_1} :\Longleftrightarrow t_{\mathsf{P}_1} \neq t^* - 1 \ \vee\ t^* \text{ is odd}$

$\mathsf{allow\text{-}bad\text{-}rand}_{\mathsf{P}_1} :\Longleftrightarrow t_{\mathsf{P}_1} \neq t^* \ \vee\ t_{\mathsf{P}_1} \neq t^* - 1$

$\mathsf{allow\text{-}corr}_{\mathsf{P}_2} :\Longleftrightarrow t_{\mathsf{P}_2} \neq t^* - 1 \ \vee\ t^* \text{ is even}$

$\mathsf{allow\text{-}bad\text{-}rand}_{\mathsf{P}_2} :\Longleftrightarrow t_{\mathsf{P}_2} \neq t^* \ \vee\ t_{\mathsf{P}_2} \neq t^* - 1$

Fig. 3: Oracles corresponding to party $\mathsf{P}_1$ of the CKA security game for a scheme $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}P}_1, \mathsf{CKA\text{-}Init\text{-}P}_2, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$; the oracles for $\mathsf{P}_2$ are defined analogously. Conditions for the weaker security game, i.e., $\varepsilon$-security, are presented to the left of those for the stronger game, i.e., $(\varepsilon, +)$-security.

epoch counters $t_{\mathsf{P}_1}$ and $t_{\mathsf{P}_2}$. Procedure **init** takes a value $t^*$, which determines in which round the challenge oracle may be called; the task of the adversary will be to recover the key $I_{t^*}$ for that round.

Upon completion of the initialization procedure, the attacker gets to interact arbitrarily with the remaining oracles, as long as *the calls are in a "ping-pong" order*, i.e., a call to a send oracle for $\mathsf{P}_1$ is followed by a receive call for $\mathsf{P}_2$, then by a send oracle for $\mathsf{P}_2$, etc. The attacker only gets to use the challenge oracle for epoch $t^*$. The attacker additionally has the capability of testing the consistency of $T_t$ and $I_t$ (i.e., whether the receiver in epoch $t$ would produce key $I_t$ on input message $T_t$).

The security game of ACD is parametrized by $\Delta_{\mathsf{CKA}}$, which stands for the number of epochs that need to pass after $t^*$ until the states do not contain secret information pertaining to the challenge. Once a party reaches epoch $t^*+\Delta_{\mathsf{CKA}}$, its state may be revealed to the attacker (via the corresponding corruption oracle). We avoid this and define two levels of security, the former weaker than the

latter. At the bottom of Figure 3, the conditions $\mathsf{allow\text{-}corr_P}$ and $\mathsf{allow\text{-}bad\text{-}rand_P}$ for the weaker version are presented to the left of those of the stronger version. We define two levels of security in order to capture a stronger, more fine-grained security guarantee for CKA which will be useful in providing stronger security guarantees for the DR and TR as a whole when one considers the composition of all its building blocks, CKA being one of them. In the former, bad randomness is not allowed in the epochs $t^*$ and $t^*-1$, and corruptions are not allowed in the epoch $t^*$ after invoking CKA-S (for the sender of epoch $t^*$) and before invoking CKA-R (for the receiver of epoch $t^*$). In the latter, which is used by the TR, bad randomness is not allowed in the epochs $t^*$ and $t^*-1$, and corruption of the receiver of epoch $t^*$ is not allowed before invoking CKA-R (for epoch $t^*$). There is no other difference between the two notions.

The game ends (not made explicit) once both states are revealed after the challenge phase. The attacker wins the game if it eventually outputs the correct secret key $I_{t^*}$ corresponding to the challenge message $T_{t^*}$. The advantage of an attacker $\mathcal{A}$ against a CKA scheme CKA is denoted by $\mathsf{Adv}^{\mathsf{CKA}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathsf{CKA}^+}(\mathcal{A})$ for the weaker and stronger security guarantees, respectively. The attacker is parameterized by its running time $t$.

**Definition 2.** *A CKA scheme* CKA *is* $(t, \varepsilon)$-secure *(resp.* $(t, \varepsilon, +)$-secure*) if for all $t$-attackers $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{CKA}}(\mathcal{A}) \leq \varepsilon \text{ (resp. } \mathsf{Adv}^{\mathsf{CKA}^+}(\mathcal{A}) \leq \varepsilon).$$

**Definition 3.** *A CKA scheme* CKA *is simply called* $\varepsilon$-secure *(resp.* $(\varepsilon, +)$-secure*) if for every $t \in \mathrm{poly}(\kappa)$ and $\varepsilon \in \mathrm{negl}(\kappa)$, where $\kappa$ is the security parameter,*

$$\mathsf{Adv}^{\mathsf{CKA}}(\mathcal{A}) \leq \varepsilon \text{ (resp. } \mathsf{Adv}^{\mathsf{CKA}^+}(\mathcal{A}) \leq \varepsilon).$$

Observe that since the TR uses a CKA with the latter, stronger security, the attack of Section 1.4 is thwarted. This is because even if the epoch $t^*$ sender is corrupted after invoking CKA-S, $I_{t^*}$ should remain hidden.

*Remark 1.* Many *natural* CKA schemes satisfy an additional property that given a CKA message $T$ and key $I$ for a given round, it is possible to deterministically compute the corresponding state of the receiving party after her execution of CKA-R in that round. We model this explicitly by requiring a deterministic algorithm CKA-Der-R that takes a message $T$ and key $I$ and produces the correct state $\gamma' \leftarrow \mathsf{CKA\text{-}Der\text{-}R}(T, I)$. All CKA schemes in this work are required to be *natural.*

### Differences from ACD

*Fine-grained security guarantees.* Recall the "CKA from DDH" scheme from ACD (which is the public ratchet used in the DR and which we prove security for in Appendix B.3), $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}P_1}, \mathsf{CKA\text{-}Init\text{-}P_2}, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$, that is instantiated in a cyclic group $G = \langle g \rangle$ as follows:

- The initial shared state $k = (h, x_0)$ consists of a (random) group element $h = g^{x_0}$ and its discrete logarithm $x_0$. The initialization for $\mathsf{P}_1$ outputs $h \leftarrow \mathsf{CKA\text{-}Init\text{-}P}_1(k)$ and that for $\mathsf{P}_2$ outputs $x_0 \leftarrow \mathsf{CKA\text{-}Init\text{-}P}_2(k)$.
- The send algorithm $\mathsf{CKA\text{-}S}$ takes as input the current state $\gamma = h$ and proceeds as follows: It
  1. chooses a random exponent $x$;
  2. computes the corresponding key $I \leftarrow h^x$;
  3. sets the CKA message to $T \leftarrow g^x$;
  4. sets the new state to $\gamma \leftarrow x$; and
  5. returns $(\gamma, T, I)$.
- The receive algorithm $\mathsf{CKA\text{-}R}$ takes as input the current state $\gamma = x$ as well as a message $T = h$ and proceeds as follows: It
  1. computes the key $I = h^x$;
  2. sets the new state to $\gamma \leftarrow h$; and
  3. returns $(\gamma, I)$.

Now, let $x_0$ be the random exponent that is part of the initial shared state, and for $i > 0$, let $x_i$ be the random exponent picked by $\mathsf{CKA\text{-}S}$ (which was run by $\mathsf{P}_1$ for odd $i$, and $\mathsf{P}_2$ for even $i$) in round $i$. Then, we have the following:

- The key for round $i$ is $I_i = g^{x_{i-1}x_i}$.
- The message for round $i$ is $T_i = g^{x_i}$.
- If $i$ is odd, and $\mathsf{P}_1$ has yet to invoke $\mathsf{CKA\text{-}S}$, $\gamma^{\mathsf{P}_1} = g^{x_{i-1}}$ and $\gamma^{\mathsf{P}_2} = x_{i-1}$.
- If $i$ is odd, and $\mathsf{P}_1$ has invoked $\mathsf{CKA\text{-}S}$, but $\mathsf{P}_2$ has yet to invoke $\mathsf{CKA\text{-}R}$, $\gamma^{\mathsf{P}_1} = x_i$ and $\gamma^{\mathsf{P}_2} = x_{i-1}$.
- If $i$ is odd, and $\mathsf{P}_1$ has invoked $\mathsf{CKA\text{-}S}$, and $\mathsf{P}_2$ has invoked $\mathsf{CKA\text{-}R}$, $\gamma^{\mathsf{P}_1} = x_i$ and $\gamma^{\mathsf{P}_2} = g^{x_i}$.
- If $i$ is even, and $\mathsf{P}_2$ has yet to invoke $\mathsf{CKA\text{-}S}$, $\gamma^{\mathsf{P}_1} = x_{i-1}$ and $\gamma^{\mathsf{P}_2} = g^{x_{i-1}}$.
- If $i$ is even, and $\mathsf{P}_2$ has invoked $\mathsf{CKA\text{-}S}$, but $\mathsf{P}_1$ has yet to invoke $\mathsf{CKA\text{-}R}$, $\gamma^{\mathsf{P}_1} = x_{i-1}$ and $\gamma^{\mathsf{P}_2} = x_i$.
- If $i$ is even, and $\mathsf{P}_2$ has invoked $\mathsf{CKA\text{-}S}$, and $\mathsf{P}_1$ has invoked $\mathsf{CKA\text{-}R}$, $\gamma^{\mathsf{P}_1} = g^{x_i}$ and $\gamma^{\mathsf{P}_2} = x_i$.

Based on the above, we make the following observations:

- If $i$ is odd and $\mathsf{P}_1$ is corrupted after invoking $\mathsf{CKA\text{-}S}$, the adversary learns $\gamma^{\mathsf{P}_1} = x_i$ and since it also has access to $g^{x_j}$ for all $j \geq 1$, the adversary learns $I_i$ and $I_{i+1}$.
- If $i$ is even and $\mathsf{P}_1$ is corrupted after invoking $\mathsf{CKA\text{-}R}$, and $\mathsf{P}_2$ used good randomness in picking $x_i$ while invoking $\mathsf{CKA\text{-}S}$ in round $i$, the adversary learns $\gamma^{\mathsf{P}_1} = g^{x_i}$, but since it only (assuming no other corruptions) has access to $g^{x_j}$ for all $j \geq 1$, the adversary does not learn $I_i$ (if $\mathsf{P}_1$ also used good randomness in picking $x_{i-1}$ while invoking $\mathsf{CKA\text{-}S}$ in round $i-1$) or $I_{i+1}$ (if $\mathsf{P}_1$ also uses good randomness in picking $x_{i+1}$ while invoking $\mathsf{CKA\text{-}S}$ in round $i+1$).

Thus, the CKA keys for two rounds are compromised only in the case where the party that has last sent a message is corrupted, and not if the party has last received a message. This allows us to consider a more fine-grained version of the CKA security game than the one described in ACD.

*Non-malleability.* Consider the following scenario in the DR or TR: It is $\mathsf{P}_1$'s turn to start a new sending epoch, but she has not yet. Then her state is leaked, and afterwards, she sends the first message $m_1$ of the epoch with good randomness. Then, if $\mathsf{P}_2$ started her last epoch with good randomness, and there are no other leakages, $m_1$ is required to remain private by $\mathcal{F}_{\mathsf{DR}}$ and $\mathcal{F}_{\mathsf{TR}}$, respectively. However, all authenticity for $m_1$ is lost—the adversary leaked on $\mathsf{P}_1$ beforehand and thus could have generated the message herself. Therefore, we replace the indistinguishability definition of ACD with our recoverability definition and require non-malleability of CKA messages via the **test** oracle—the adversary should not be able to maul them in order to learn about the actual session messages sent in the DR or TR. Note that this modification makes our CKA definition incomparable in strength to that of ACD, but allows us to prove stronger security for the DR. See the full security proof of Theorem 5 for the DR and TR, as well as Appendix D.2, for more details.

**Instantiating $\mathsf{CKA}^+$** A CKA scheme $\mathsf{CKA}^+ = (\mathsf{CKA\text{-}Init\text{-}P_1}, \mathsf{CKA\text{-}Init\text{-}P_2}, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$ (which may be used in the TR) can be obtained assuming random oracles or circular-secure ElGamal in a cyclic group $G = \langle g \rangle$ (with exponent space $\mathcal{X}$) using a function $\mathsf{H} : \mathcal{I} \to \mathcal{X}$ as follows:

- The initial shared state $k = (h, x_0)$ consists of a (random) group element $h = g^{x_0}$ and its discrete logarithm $x_0$. The initialization for $\mathsf{P}_1$ outputs $h \leftarrow \mathsf{CKA\text{-}Init\text{-}P_1}(k)$ and that for $\mathsf{P}_2$ outputs $x_0 \leftarrow \mathsf{CKA\text{-}Init\text{-}P_2}(k)$.
- The send algorithm CKA-S takes as input the current state $\gamma = h$ and proceeds as follows: It
  1. chooses a random exponent $x$;
  2. computes the corresponding key $I \leftarrow h^x$;
  3. sets the CKA message to $T \leftarrow g^x$;
  4. sets the new state to $\gamma \leftarrow x \cdot \mathsf{H}(I)$; and
  5. returns $(\gamma, T, I)$.
- The receive algorithm CKA-R takes as input the current state $\gamma = x$ as well as a message $T = h$ and proceeds as follows: It
  1. computes the key $I = h^x$;
  2. sets the new state to $\gamma \leftarrow h^{\mathsf{H}(I)}$; and
  3. returns $(\gamma, I)$.

Note that the above scheme is *natural*, i.e., it supports a CKA-Der-R algorithm, namely, $\mathsf{CKA\text{-}Der\text{-}R}(T, I) = T^{\mathsf{H}(I)}$. Now we show its security in the theorem below. (We give informal details on additional security properties that we conjecture it to have in Appendix B.3)

**Theorem 1.** *Assume group $G$ is $(t, \varepsilon)$-StDH-secure. Additionally, assume the existence of a random oracle* $\mathsf{H}$. *Then, the above CKA scheme* $\mathsf{CKA}$ *is $(t', \varepsilon, +)$-secure for $t \approx t'$.*

*Proof.* Assume w.l.o.g. that $t^*$ is *odd*, i.e., $\mathsf{P}_1$ sends the challenge; the case where $t^*$ is even is handled analogously. Let $g^a, g^b$ be a StDH challenge. The reduction simulates the CKA protocol in the straight-forward way but embeds the challenge into the CKA as follows:

- in epoch $t^* - 1$, it uses $T_{t^*-1} = g^a$ and $I_{t^*-1} = g^{xa\mathsf{H}(I_{t^*-2})}$, where $x$ is the exponent used to simulate $T_{t^*-2} = g^x$.
- in epoch $t^*$, it uses $T_{t^*} = g^b$ and $I_{t^*} = g^{ab\mathsf{H}(I_{t^*-1})}$ which is the key the adversary is to recover, as well as sets $\gamma_{t^*}^{\mathsf{P}_1} \leftarrow y$, for random $y$ in $\mathcal{X}$;
- in epoch $t^* + 1$, for exponent $x'$ (possibly generated using adversarial randomness), it uses $T_{t^*+1} = g^{x'}$ and $I_{t^*+1} = g^{yx'}$.

It is easy to verify that this correctly simulates the CKA experiment since $\mathsf{H}$ is a random oracle. In particular, randomly sampled $y$ properly simulates $b \cdot \mathsf{H}(I_{t^*})$: If $\mathcal{A}$ does not query the random oracle on $I_{t^*}$ then $y$ is properly distributed. Moreover, when $\mathcal{A}$ makes a random oracle query for any $I$, the reduction can query oracle $\mathbf{ddh}(g^a, \cdot, \cdot)$ on $(g^{b\mathsf{H}(I_{t^*-1})}, I)$ so that if indeed $I = I_{t^*}$, the reduction will know, and then forward to its challenger $g^{ab} = I_{t^*}^{1/\mathsf{H}(I_{t^*-1})}$ before answering the $\mathsf{CKA}^+$ attacker's query.

Similarly, the test oracle can be perfectly simulated with the help of $\mathbf{ddh}()$: if $\mathbf{test}(t^*, T, I)$ is queried, the reduction simply queries $\mathbf{ddh}(g^a, \cdot, \cdot)$ on $(T^{\mathsf{H}(I_{t^*-1})}, I)$; all other $\mathbf{test}()$ queries can be directly simulated. □

## Acknowledgements

We would like to thank Yevgeniy Dodis and Daniel Jost for helping us realize that the trick used in the $\mathsf{CKA}^+$ construction can also be used to make UPKE more efficient (Appendix F).

This research is supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

## References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg, Germany, San Francisco, CA, USA (Apr 8–12, 2001)
2. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. Cryptology ePrint Archive, Report 2018/1037 (2018), https://eprint.iacr.org/2018/1037
3. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 129–158. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)
4. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 248–277. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020)

5. Balli, F., Rösler, P., Vaudenay, S.: Determining the core primitive for optimally secure ratcheting. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 621–650. Springer, Heidelberg, Germany, Daejeon, South Korea (Dec 7–11, 2020)
6. Bao, F., Deng, R.H., Zhu, H.: Variations of diffie-hellman problem. In: ICICS (2003)
7. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: The security of messaging. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 619–650. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
8. Bienstock, A., Fairoze, J., Garg, S., Mukherjee, P., Srinivasan, R.: A more complete analysis of the signal double ratchet algorithm. Cryptology ePrint Archive, Report 2022/355 (2022)
9. Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 266–284. Springer, Heidelberg, Germany, Taormina, Sicily, Italy (Mar 19–21, 2012)
10. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record communication, or, why not to use pgp. In: Proceedings of the 2004 ACM workshop on Privacy in the electronic society. pp. 77–84 (2004)
11. Brendel, J., Fiedler, R., Günther, F., Janson, C., Stebila, D.: Post-quantum asynchronous deniable key exchange and the signal handshake. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography – PKC 2022. pp. 3–34. Springer International Publishing, Cham (2022)
12. Brendel, J., Fischlin, M., Günther, F., Janson, C.: PRF-ODH: Relations, instantiations, and impossibility results. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 651–681. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
13. Brendel, J., Fischlin, M., Günther, F., Janson, C., Stebila, D.: Towards post-quantum security for signal's x3dh handshake. In: Selected Areas in Cryptography–SAC 2020 (2020)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001)
15. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg, Germany, Interlaken, Switzerland (May 2–6, 2004)
16. Canetti, R., Jain, P., Swanberg, M., Varia, M.: Universally composable end-to-end secure messaging. In: CRYPTO 2022 (2022)
17. Checkoway, S., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D.J., Maskiewicz, J., Shacham, H., Fredrikson, M.: On the practical exploitability of dual EC in TLS implementations. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 319–335. USENIX Association, San Diego, CA, USA (Aug 20–22, 2014)
18. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. pp. 451–466. IEEE (2017), https://doi.org/10.1109/EuroSP.2017.27
19. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. J. Cryptol. 33(4), 1914–1983 (2020), https://doi.org/10.1007/s00145-020-09360-1

20. Cohn-Gordon, K., Cremers, C.J.F., Garratt, L.: On post-compromise security. In: Hicks, M., KÃPpf, B. (eds.) CSF 2016 Computer Security Foundations Symposium. pp. 164–178. IEEE Computer Society Press, Lisbon, Portugal (jun 27-1 2016)
21. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing 33(1), 167–226 (Nov 2003)
22. Dobson, S., Galbraith, S.D.: Post-quantum signal key agreement with sidh. Cryptology ePrint Archive, Report 2021/1187 (2021)
23. Dodis, Y., Karthikeyan, H., Wichs, D.: Updatable public key encryption in the standard model (2021)
24. Durak, F.B., Vaudenay, S.: Breaking the FF3 format-preserving encryption standard over small domains. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 679–707. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
25. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. In: Attrapadung, N., Yagi, T. (eds.) IWSEC 19. LNCS, vol. 11689, pp. 343–362. Springer, Heidelberg, Germany, Tokyo, Japan (Aug 28–30, 2019)
26. FIPS, P.: 180-1. secure hash standard. National Institute of Standards and Technology 17,  45 (1995)
27. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press (2012)
28. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004), http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html
29. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989), https://doi.org/10.1137/0218012
30. Hashimoto, K., Katsumata, S., Kwiatkowski, K., Prest, T.: An efficient and generic construction for signal's handshake (x3dh): Post-quantum, state leakage secure, and deniable. In: Public Key Cryptography (2). pp. 410–440 (2021)
31. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your ps and qs: Detection of widespread weak keys in network devices. In: Kohno, T. (ed.) USENIX Security 2012. pp. 205–220. USENIX Association, Bellevue, WA, USA (Aug 8–10, 2012)
32. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: The safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 33–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
33. Jost, D., Maurer, U.: Overcoming impossibility results in composable security using interval-wise guarantees. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 33–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020)
34. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: Almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 159–188. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)
35. Jost, D., Maurer, U., Mularczyk, M.: A unified and composable take on ratcheting. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 180–210. Springer, Heidelberg, Germany, Nuremberg, Germany (Dec 1–5, 2019)

36. Kiltz, E.: A tool box of cryptographic functions related to the Diffie-Hellman function. In: Rangan, C.P., Ding, C. (eds.) INDOCRYPT 2001. LNCS, vol. 2247, pp. 339–350. Springer, Heidelberg, Germany, Chennai, India (Dec 16–20, 2001)
37. Kobeissi, N., Bhargavan, K., Blanchet, B.: Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 435–450 (2017)
38. Krawczyk, H., Eronen, P.: Hmac-based extract-and-expand key derivation function (hkdf). Tech. rep., RFC 5869, May (2010)
39. Kurosawa, K., Matsuo, T.: How to remove MAC from DHIES. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 04. LNCS, vol. 3108, pp. 236–247. Springer, Heidelberg, Germany, Sydney, NSW, Australia (Jul 13–15, 2004)
40. Marlinspike, M., Perrin, T.: The Double Ratchet Algorithm (11 2016), https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf
41. Marlinspike, M., Perrin, T.: The X3DH Key Agreement Protocol (11 2016), https://signal.org/docs/specifications/x3dh/x3dh.pdf
42. Maurer, U.: Constructive cryptography–a new paradigm for security definitions and proofs. In: Joint Workshop on Theory of Security and Applications. pp. 33–56. Springer (2011)
43. Maurer, U.M., Wolf, S.: Diffie-Hellman oracles. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 268–282. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996)
44. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002)
45. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 3–32. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
46. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997)
47. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company (1997)
48. Unger, N., Goldberg, I.: Deniable key exchanges for secure messaging. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1211–1223. ACM Press, Denver, CO, USA (Oct 12–16, 2015)
49. Unger, N., Goldberg, I.: Improved strongly deniable authenticated key exchanges for secure messaging. Proc. Priv. Enhancing Technol. 2018(1), 21–66 (2018)
50. Vatandas, N., Gennaro, R., Ithurburn, B., Krawczyk, H.: On the cryptographic deniability of the Signal protocol. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) ACNS 20, Part II. LNCS, vol. 12147, pp. 188–209. Springer, Heidelberg, Germany, Rome, Italy (Oct 19–22, 2020)
51. Yilek, S., Rescorla, E., Shacham, H., Enright, B., Savage, S.: When private keys are public: Results from the 2008 debian openssl vulnerability. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement. pp. 15–27. IMC '09, Association for Computing Machinery, New York, NY, USA (2009), https://doi.org/10.1145/1644893.1644896