Stronger Secure Messaging with Immediate Decryption and Constant-Size Overhead

No Institute Given

Abstract. We further explore the limits of the security that can be achieved by modern secure messaging protocols that satisfy real-world requirements such as immediate decryption – the ability to decrypt messages as they arrive, even if received out-of-order or if some messages never arrive – and constant-size message overhead. With respect to immediate decryption, the crucial part of such protocols is the continuous phase after the initial key establishment, which deals with asynchronous message sending and receiving messages out-of-order. Until now, the strongest known provably-secure construction in this space is the ACD19 protocol that meets the so-called Secure-Messaging (SM) security notion. In this work, we propose an extended-Secure-Messaging (eSM) security notion that achieves even stronger fine-grained security, for example against the compromise of session-specific state. Unlike previous attempts to strengthen the security of secure messaging, this notion can be achieved while retaining immediate decryption and constant-size overhead. To show this, we propose such a protocol construction that provably achieves eSM security. Our protocol is the strongest secure proposed to date with immediate decryption and constant-size overhead. We additionally show how our protocol can be combined with the SPQR protocol to form a complete protocol that is PQ-secure and offline-deniable.

Keywords: Provable Security, Secure messaging, Double Ratchet, Immediate Decryption, Stronger Security, Compromise, Deniability.

1 Introduction

The Signal protocol for secure messaging, and in particular its underlying Signal protocol library, have seen global uptake in the last few years and have brought strong security guarantees to many messaging applications worldwide. The core design that underlies Signal can be viewed as consisting of two main building blocks: an initial X3DH asynchronous key exchange, and the double-ratchet that evolves the state of both parties during the subsequent message exchanges. During the last years, both of these building blocks have seen much development in the details of their analysis, and the possible strengthenings of the formal security guarantees that could theoretically be achieved.

To achieve very strong security guarantees in the Double Ratchet message exchange phase, keys are evolved with each sent message, possibly involving new randomness that gets integrated into key chains; ideally, to achieve the strongest security, a message key depends on all preceding key material, as well as new material. To decrypt a message, the recipient follows the same key evolution process that the sender performed. However, this implies that a message can only be decrypted once all previous messages have been received and processed.

In reality, messages can get lost, or arrive out-of-order. If the sender sends a sequence of messages, the recipient would like to be able to decrypt message i, even if message j ($j < i$) arrives only later or never at all. The ability of the recipient to decrypt a message directly when it arrives, irrespective of the arrival of prior messages, is called *immediate decryption*. The main work that studied this is Alwen et al. [1], who proposed a generalization of Signal's security notion and explicitly considered immediate decryption.

Concurrently, many works such as [2]–[5] have proposed stronger security notions for secure messaging and notably its message exchanging phase, but these designs did not explicitly consider immedate decryption as a goal, and their designs in general do not offer this feature, effectively sacrificing it to achieve stronger security notions. In theory, these can be reconciled: immediate decryption can be achieved for any secure messaging protocol if the sender simply includes all preceding ciphertexts when sending the next message. Thus, when sending message i, the sender also includes the ciphertexts for all messages $j < i$. However, this solution is not useful in practice, since this solution implies that the size of the overhead grows linearly over time. This is impractical, and such solutions are not deployed in practice.

In this work we go back to the goal of providing stronger security while explicitly including the requirement of immediate decryption with constant message overhead. We show that it is in fact possible to achieve stronger security under these constraints. Similar to the designs in [1], our protocol can be instantiated with post-quantum secure primitives.

Our goal is to cover the main security aspects of our design in comparison to, e.g., Signal. One of the design constraints for Signal was to achieve a form of deniability. Thus, we also set out to prove offline-deniability for our protocol. However, this is more challenging than perhaps expected, because we would like to also prove this for the post-quantum secure setting, which means we cannot directly rely on previous approaches.

Contributions. Our main contribution is that we provide an extended Double Ratchet protocol, which offers stronger security than ACD19 and other SM protocols while retaining immediate decryption, post-quantum compatibility, and constant-size message overhead. We introduce a related new stronger security notion called Extended-Secure-Messaging (eSM). We show that the eSM notion is stronger than other proposals with the same constraints, and prove that our protocol meets it.

Furthermore, to show offline deniability of our protocol even when instantiated in the post-quantum setting, we extend the game-based definition from SPQR [6] to the multi-stage setting. We prove that the combination of our eSM-secure protocol and SPQR (currently the only provably secure PQ-asynchronous key establishment), is offline deniable.

Overview. We give background and related work in Section 2, and preliminaries in Section 3. We propose our new eSM syntax and security notion in Section 4. We propose our concrete protocol that is provably eSM-secure in Section 5, and show its offline-deniability when combined with SPQR in Section 6.

We provide the full proofs of our theorems in the supplementary materials.

2 Background and Related Work

2.1 Instant Messaging Protocols and Immediate Decryption

The Signal protocol provably offers strong security guarantees, such as *forward secrecy* and *post-compromise security*, as proven by Cohn-Gordon et al. [7], [8], and *offline deniability*, as proven by [9]. Moreover, Signal has several features that are criticial for large-scale real-world deployment, such as *message-loss resilience* and *immediate decryption.* Roughly speaking, message-loss resilience and immediate decryption enable the receiver to decrypt a legitimate message immediately after it is received, even when some messages arrive out-of-order or are permanently lost by the network. Furthermore, the Signal protocol provides the above properties with constant message size overhead.

The core Signal protocol consists of two components: the *Extended Triple Diffie-Hellman* (X3DH) and the *Double Ratchet* (DR) framework. The X3DH is an asynchronous key establishment protocol that covers the key registration of each party and the initialization of the shared state in each communication channel; the Double Ratchet is used for the subsequent message transmission and state update in each communication channel.

Alwen et al. [1] introduced the notion of *Secure Messaging* (SM), which is a syntax and associated security notion that generalizes the security of Signal's Double Ratchet. The SM notion is further modularized into three building blocks: the *Continuous Key Agreement* (CKA), where the sender updates the shared state using its randomness, which provides the post-compromise security; the *Forward-Secure Authenticated Encryption with Associated Data* (FS-AEAD), where the sender sends messages to the recipient and updates the shared state in a deterministic manner, which provides forward secrecy and immediate decryption; the PRF-PRNG serves for the joint that inputs the output of CKA and refreshes the input of FS-AEAD. Alwen et al. also provide a concrete instantiation and prove that it is SM-secure. This instantiation is not explicitly named in [1]: in this work, we will refer to their SM-secure construction as ACD19.

A potential concern with the ACD19 protocol is that each message is deterministically encrypted using the state shared by both parties. The corruption of the shared state of either party immediately compromises the subsequent messages in a sequence and entails the loss of fine-grained security. To mitigate the impact of the state exposure, Alwen et al. [1] also propose a second security notion for secure messaging, called PKSM, and a corresponding PKSM-secure construction, which we will call ACD19-PK. Compared to ACD19, ACD19-PK additionally employs public key encryption (PKE) and digital signature (DS) schemes. In parallel to the CKA phase, the sender in ACD19-PK additionally samples two key pairs: one from PKE for decrypting next messages from the partner, and the other from DS for signing messages after a back-and-forth round. In parallel to the FS-AEAD phase, the sender additionally encrypts the ciphertext of FS-AEAD by using PKE under the recipient's public key, followed by signing the ciphertext of PKE and the newly sampled public keys under its own current DS private signing key. Intuitively, ACD19-PK provides the improved resilience to the state exposure: the compromise of the message is prevented under the state exposure from the sender side, since the attacker cannot recover the ciphertext of FS-AEAD from the ciphertext PKE without knowing the recipient's decryption key. However, the main focus of [1] are SM and ACD19: for ACD19-PK, neither a formal security claim not a concrete proof is given; thus, its additional security is essentially conjectured.

In a parallel line of research, several messaging protocols have been proposed to meet all manner of security guarantees, such as "optimal" [2], [3] and "sub-optimal" [5]. Although they follow different ratcheting frameworks aiming at various flavors of security, as we review their designs in Supplementary Material A, none of them provides immediate decryption in a natural way, due to their key-updatable or state-updatable structure. Of course, one can unnaturally impose the immediate decryption upon these protocols by attaching the past transcript to the next message transmission. However, this modification entails linearly growing bandwidth, contrary to the real-world usability and efficiency requirements. Thus, this approach has not found its way to any deployed protocols.

2.2 Offline Deniability and Post-Quantum Security

The property of *offline deniability* prevents a judge from deciding whether an honest user has participated in a conversation even when other participants try to frame them. Historically, offline deniability for the authenticated key exchange (AKE) was first defined by Di Raimondo et al. [10] in the simulation-based paradigm. Following this notion, the offline deniability of a series of AKE protocols were analyzed in [9], [10], such as MQV, HMQV, 3DH, and X3DH.

Afterwards, [9] extended the simulation-based offline deniability to full messaging protocols and studied the relation to the one of AKE: if the transcript after the AKE solely depends on the initial secret shared by both parties from AKE, the public keys, and each party's private inputs, then the full messaging protocol is offline deniable if the AKE is offline deniable. In particular, this shows that the Signal protocol achieves offline deniability in the simulation-based paradigm.

While the analysis for the offline deniability of messaging protocols in the classical setting has been well-studied, the offline deniability of messaging protocols in the post-quantum setting is surprisingly complicated. Hashimoto et al. [11] propose the first post-quantum secure key establishment but unfortunately have to restrict the offline deniability analysis to the classical setting. The main obstacle for applying their proof to the post-quantum setting is that their proof requires a strong knowledge-type assumption (i.e., plaintext-awareness for KEM), for which it is unknown whether it holds against quantum attackers. To fill the missing gap for the offline deniability of AKE in the post-quantum setting, a subsequent work by Brendel et al. [6] proposed their post-quantum asynchronous deniable key exchange (DAKE) protocol, called SPQR, and a new game-based offline deniability notion. Brendel et al. proved that their SPQR protocol is offline deniable in the game-based paradigm even against quantum attackers.

To the best of our knowledge, to date there is no definition for the offline deniability of full messaging protocols in the game-based paradigm. Although the combination of the post-quantum variants of X3DH, such as SPQR, and the post-quantum compatible ACD19 or ACD19-PK, *does* provide the promising privacy and authenticity in the post-quantum setting, it is still an *open* question, what flavors of offline deniability can be obtained for the combined protocols in the post-quantum setting.

3 Preliminaries

Notation: We assume that each algorithm A has a security parameter λ and a public parameter pp as implicit inputs. In this paper, all algorithms are executed in polynomial time. For any positive integer n, let $[n] := \{1, ..., n\}$ denote the set of integers from 1 to n. For a deterministic algorithm A, we write $y \leftarrow A(x)$ for running A with input x and assigning the output to y. Analogously, we write $y \overset{\$}{\leftarrow} A(x; r)$ for a probabilistic algorithm A using randomness r, which is sometimes omitted when it is irrelevant. We write $[\![\cdot]\!]$ for a boolean statement that is either true (denoted by 1) or false (denoted by 0). We define an event symbol $\perp$ that does not belong to any set in this paper. Let $(\cdot)$ and $\{\cdot\}$ respectively denote an ordered tuple and an unordered set. Let $n{+}{+}$ denote the increment of number n by 1, i.e., $n \leftarrow n + 1$. We use $\_$ to denote a value that is irrelevant. We use $\mathcal{D}$ to denote a dictionary that stores values for each index. The initialization of the dictionary is denoted by $\mathcal{D}[\cdot] \leftarrow \perp$. In this paper, we use **req** to indicate that a (following) condition is required to be true. If the following condition is false, then the algorithm or oracle containing this keyword is exited and all actions in this invocation are undone.

We recall the relevant cryptographic primitives in Supplementary Material B.

4 Extended Secure Messaging

In this section we present our stronger security notion eSM that strengthens the secure messaging (SM) security notion from [1].

The SM notion was defined communication phase that follows the initial key establishment, and only made use of the shared secret from that initial phase. One of its instantiations, the ACD19 protocol, provably provides various security guarantees, including immediate decryption. However, their design does not achieve a fine-grained notion of security. For instance, the corruption of either the sender's or the receiver's shared state compromises the subsequent messages. Another instantiation in [1], ACD19-PK, was conjectured to meet a more fine-grained notion of security, but this was not formally defined nor proven. It is natural to ask whether even stronger security can be provably achieved.

A key observation is that the SM scheme only inputs the shared secret from the initial key establishment; in contrast, the popular designs of the initial key establishment phase, such as the $\mathsf{X3DH}$ and SPQR, obtain substantial benefits from the usage of the long-term identity keys and the medium-term prekeys. Intuitively, we will re-use the identity keys and prekeys in the subsequent message exchange phase to obtain additional security guarantees.

We first introduce the definition of the *extended secure messaging* scheme in Section 4.1, followed by the expected security properties in Section 4.2. Finally, we define our strong messaging security model (eSM) in Section 4.3 and explain how this model captures the intended security properties in Section 4.4.

4.1 Syntax

Definition 1. *Let $\mathcal{ISS}$ denote the space of the initial shared secrets between two parties. An* extended secure messaging *(eSM) scheme consists of six algorithms* $\mathsf{eSM} = (\mathsf{IdKGen}, \mathsf{PreKGen}, \mathsf{eInit\text{-}A}, \mathsf{eInit\text{-}B}, \mathsf{eSend}, \mathsf{eRcv})$*, where*

- IdKGen *outputs an identity public-private key pair* $(ipk, ik) \xleftarrow{\$} \mathsf{IdKGen}()$
- $\mathsf{PreKGen}$ *outputs a medium-term pre-key pair* $(prepk, prek) \xleftarrow{\$} \mathsf{PreKGen}()$
- $\mathsf{eInit\text{-}A}$ *(resp.* $\mathsf{eInit\text{-}B}$*) inputs an initial shared secret* $iss \in \mathcal{ISS}$ *and outputs a state* $\mathsf{st}_{\mathtt{A}} \xleftarrow{\$} \mathsf{eInit\text{-}A}(iss)$ *(resp.* $\mathsf{st}_{\mathtt{B}} \xleftarrow{\$} \mathsf{eInit\text{-}B}(iss)$*).*
- eSend *inputs a state st, a long-term identity public key ipk, a medium-term public prekey prepk, and a message m, and outputs a new state and a ciphertext* $(st', c) \xleftarrow{\$} \mathsf{eSend}(st, ipk, prepk, m)$*, and*
- eRcv *inputs a state st, a long-term identity private key ik, a medium-term private key prek, and a ciphertext c, and outputs a new state, an epoch number, a message index, and a message* $(st', t, i, m) \leftarrow \mathsf{eRcv}(st, ik, prek, c)$*.*

Two important concepts for eSM are the epoch number and message index.

Epoch. The epoch t is used to describe how far the interaction between two parties in a session has proceeded. Let $t_{\mathtt{A}}$ and $t_{\mathtt{B}}$ respectively denote the epoch counter of parties $\mathtt{A}$ and $\mathtt{B}$ in a session. Both epoch counters start from 0. If either party $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ switches the actions, i.e., from sending to receiving or from receiving to sending messages, then the corresponding counter $t_{\mathtt{P}}$ is incremented by 1. Throughout this paper, we use even epochs ($t_{\mathtt{A}}, t_{\mathtt{B}} = 0, 2, 4, ...$) to denote the scenario where $\mathtt{B}$ acts as the message sender $\mathtt{A}$ acts as the message receiver, and odd epochs in reverse. In each epoch, the sender can send an arbitrary number of messages in a sequence. The difference between the two counters $t_{\mathtt{A}}$ and $t_{\mathtt{B}}$ is never greater than 1, i.e., $|t_{\mathtt{A}} - t_{\mathtt{B}}| \leq 1$.

Message Indices. The message index i identifies the position of a message in each epoch, in particular, when the messages in a sequence are (possibly) delivered out of order. Notably, the epoch number t and message index i output by eRcv indicate the position of the decrypted message m during the communication.

Syntax extensions. Compared to the original SM syntax definition from [1], eSM has two additional algorithms IdKGen and PreKGen: IdKGen outputs the public-private identity key, which is fixed once generated, and PreKGen outputs pre-key pairs, which are updated regularly (similar to X3DH).

The eInit-A and eInit-B respectively initialize the session-specific states $\mathsf{st}_\mathtt{A}$ and $\mathsf{st}_\mathtt{B}$ of parties A and B using the initial shared secret $iss \in \mathcal{ISS}$, which is assumed to be produced by a key establishment.

We assume that all the session-specific data is stored at the same security level in the state st, but that data shared among multiple sessions (such as identity keys and pre-keys) may be stored differently. Even if all data is stored at the same security level, we still assume that the compromise of some data of a party does not automatically imply the compromise of all other data of that party – this assumption is in fact the same that yields the added benefits of the Naxos and X3DH protocol designs. In fact, as we will show later in Section 4.4, an eSM scheme can achieve additional privacy guarantees if the private identity keys (or pre-keys) can be stored in the secure environment on the device, such as Hardware Security Module (HSM).

4.2 Strong Security Properties

The eSM schemes aim to satisfy following strong security properties. First, we expect eSM to preserve the basic security properties defined for SM in [1]:

- **Correctness:** The messages sent by one party and delivered to the other should be recovered in the correct order, if the attacker interferes with the underlying transmissions.
- **Immediate decryption and message-loss resilience (MLR):** Messages must be decrypted as soon as they arrive; the loss of some messages does not prevent subsequent interaction.
- **Forward secrecy (FS)**: All messages sent and received prior to a session state compromise of either party (or both) remain confidential to an attacker.
- **Post-compromise security (PCS):** The parties can recover from a session state compromise (assuming each has access to fresh randomness) when the attacker is passive.

Additionally, we note that eSM also targets the following stronger security properties than SM in [1].

- ***Strong* authenticity:** The attacker cannot modify the messages in transmission or inject new ones, unless the sender's session state is compromised.
- ***Strong* privacy**: If both parties' states are uncompromised, the attacker obtains no information about the messages sent. Assuming both parties have access to fresh randomness, strong privacy also holds unless the receiver's session state, private identity key, and private pre-key all are compromised.
- **Randomness leakage/failures**: While both parties' session states are uncompromised, all the security properties above (in particular, including strong authenticity and strong privacy) except PCS hold even if the attacker completely controls the parties' local randomness. That is, good randomness is only required for PCS.

In [1], the authenticity and privacy hold only when neither parties' states are compromised. Instead, we aim for stronger authenticity and strong privacy, which respectively allow the compromise of the receiver's and the sender's state. This also implies stronger security against randomness leakage/failures.

Finally, our eSM also pursues two more novel security properties:

– ***State compromise/failures*:** While the sender's randomness is unpredictable and the receiver's private identity key or pre-key is uncompromised, the privacy of the messages holds even if both parties' session states are corrupted. This means that the confidentiality of the messages does not solely rely on the privacy of the session states.
– ***Periodic confidentiality recovery* (PCR):** If the attacker is passive (i.e., does not inject corrupt messages), the message confidentiality recovers from the compromise of both parties' all private information after a period (assuming each has access to fresh randomness).

The first new property is *state compromise/failures*, which enables the confidentiality even under the compromise of both parties' session states. We stress that this property has a particular impact for the secure messaging after an *insecure* key establishment. For instance, consider that the party B initializes a session with A using X3DH or SPQR. The leakage of B's private identity key and ephemeral randomness implies the compromise of the initial shared secret. The surprise here is that apparently, from the initial shared secret, the attacker can learn both parties' session states in every SM scheme, such as ACD19 and ACD19-PK. If B continuously sends messages to A without receiving a reply using double ratchet in Signal protocol, or ACD19 or ACD19-PK constructions in [1], all messages in the sequence are leaked, since the attacker can use A's session state to decrypt the ciphertext. An eSM protocol with the "state compromise/failures" property is able to prevent such an attack.

The second one is *Periodic Confidentiality Recovery* (PCR), which enables the recovery of the message privacy even when all private data of both parties' are compromised. We stress that this property has a particular impact for practical messaging protocols with immediate decryption if the parties are communicating over unauthenticated channels, which is the common scenario in real life. Suppose that the private data of A is leaked and B is continuously sending messages to A. The compromise of the messages from B to A right after A's private data corruption is unavoidable, since the attacker can perform any step that A can do, in order to decrypt the ciphertext. The party A might want to send a reply to B using fresh randomness to heal the state (ensured by PCS) and then the privacy of the subsequent messages (ensured by strong privacy). However, the attacker can intercept all subsequent messages from A to B over the unauthenticated channel. Due to the immediate decryption requirement, A must be able to recover all messages from B to A even though the message sent by A is not delivered to B. This means that the attacker can also recover the messages from B to A as before. The PCR property prevents the above attack.

We can consider PCR as a complement of PCS and strong privacy: if a message produced by a fully corrupted party using fresh randomness is delivered to the partner, the confidentiality of the subsequent messages can be recovered; if such message is not delivered, the confidentiality of the subsequent messages can still be recovered after a certain period, i.e., when the corrupted party uploads new pre-keys that are generated using fresh randomness. To the best of our knowledge, the PCR property is not satisfied by any known messaging protocol.

4.3 Security Model

The *Extended Secure Messaging* (eSM) security game $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ is depicted in Figure 1. We start by explaining the notation.

Notation. For a party $\mathtt{P} \in \{\mathtt{A},\mathtt{B}\}$, we use $\neg\mathtt{P}$ to denote the partner, i.e., $\{\mathtt{P},\neg\mathtt{P}\} = \{\mathtt{A},\mathtt{B}\}$. For an element x and a set X, we write $X \overset{+}{\leftarrow} x$ for adding x into X, i.e., $X \overset{+}{\leftarrow} x \Leftrightarrow X \leftarrow X \cup \{x\}$. Similarly, we write $X \overset{-}{\leftarrow} x$ for removing x from X, i.e., $X \overset{-}{\leftarrow} x \Leftrightarrow X \leftarrow X \setminus \{x\}$. For a set of tuples X and a variable y, we use $X(y)$ to denote the subset of X, where each element includes y, i.e., $X(y) := \{x \in X \mid y \in x\}$. We sometimes write $y \in X$ to denote that there exists a tuple $x \in X$ such that $y \in x$, i.e., $y \in X \Leftrightarrow X(y) \neq \emptyset$.

Trust Model: We assume no forgery of the public identity keys and pre-keys. This is the common treatment in the security analyses in this domain, e.g. [7], the server is considered to be a bulletin board, where each party can upload their own honest public keys and fetch other parties' honest public keys. We omit the discussion on how frequent the medium-term pre-keys are generated and simply assume that the public identity keys and pre-keys can be delivered to the partner, once the keys are generated. For example, we can consider a scenario where every party is only allowed to upload and fetch public keys at 12am every day. In practice, the local data for decrypting a message that is lost for a long time will be deleted depending on storage restrictions. However, we assume sufficient storage on each device in order to capture the full immediate decryption analysis, similar to the assumption made in [1]. Moreover, we also assume that the eSM is *natural*, which is first defined for SM in [1, Definition 7].

Definition 2. *We say an* eSM *scheme is* natural*, if the following holds:*
1. *the receiver state remains unchanged, if the message output by* eRcv *is* $m = \perp$*,*
2. *the values* (t,i) *output by* eRcv *can be efficiently computed from* c*,*
3. *if* eRcv *has already accepted an ciphertext corresponding to the position* (t,i)*, the next ciphertext corresponding to the same position is rejected immediately,*
4. *a party always rejects ciphertexts corresponding to an epoch in which the party does not act as receiver, and*
5. *if a party* $\mathtt{P}$ *accepts a ciphertext corresponding to an epoch* t*, then* $t \leq t_{\mathtt{P}} + 1$*.*

Experiment Variables and Predicates. The security experiment $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ includes the following global variables:

- $\mathsf{safe}_{\mathtt{A}}^{\mathsf{idK}}$, $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}} \in \{\text{true}, \text{false}\}$: the boolean values indicating whether the attacker reveals the private identity keys.
- $\mathcal{L}_{\mathtt{A}}^{\mathsf{rev}}, \mathcal{L}_{\mathtt{B}}^{\mathsf{rev}}$: the lists that record the indices of the pre-keys that are revealed.
- $\mathcal{L}_{\mathtt{A}}^{\mathsf{cor}}, \mathcal{L}_{\mathtt{B}}^{\mathsf{cor}}$: the lists that record the indices of the epochs that are corrupted.
- $(n_{\mathtt{A}}, n_{\mathtt{B}})$: the counters that count how many pre-keys are generated.
- $(t_{\mathtt{A}}, t_{\mathtt{B}})$: the epoch counters.
- $(i_{\mathtt{A}}, i_{\mathtt{B}})$: the message index counters.
- trans: a set that records all ciphertexts, which are honestly encrypted but not delivered yet, and their related information, such as the sender identity, the receiver's pre-key index, the randomness quality during the ciphertext generation, the corresponding epoch and message index, and the encrypted message. See the helping function **record** for more details.
- $\mathsf{allTrans}$: a set that records all ciphertext, which are honestly encrypted, and their related information.
- chall: a set that records all challenge ciphertexts, which are honestly encrypted but not delivered yet, and their related information.
- $\mathsf{allChall}$: a set that records all challenge ciphertexts (including both the delivered and undelivered ones), and their related information.
- comp: a set that records all compromised ciphertexts, which are honestly encrypted but not delivered yet, and their related information. A compromised ciphertext means that the attacker can trivially forge a new ciphertext at the same position.
- $\mathsf{win}^{\mathsf{corr}}$, $\mathsf{win}^{\mathsf{auth}}$, $\mathsf{win}^{\mathsf{priv}} \in \{\text{true}, \text{false}\}$: the winning predicate that indicates whether the attacker wins.
- $\mathsf{b} \in \{0, 1\}$: the challenge bit.

Compared to [1], there are two major differences in the experiment variables. On the one hand, our model involves more variables that are related to the identity keys and pre-keys, which are not included in [1], such as $\mathsf{safe}_{\mathtt{P}}^{\mathsf{idK}}$, $\mathcal{L}_{\mathtt{P}}^{\mathsf{rev}}$, and $n_{\mathtt{P}}$, for $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$. We also import two new sets $\mathsf{allTrans}$ and $\mathsf{allChall}$ to simplify the security analysis of the benefits obtained from using the identity keys and pre-keys. On the other hand, we use two lists $\mathcal{L}_{\mathtt{A}}^{\mathsf{cor}}$ and $\mathcal{L}_{\mathtt{B}}^{\mathsf{cor}}$ to capture the state corruption of either party instead of using a single counter. While splitting the single state corruption variable into two helps our model to capture our strong privacy and strong authentication, using lists but not a counter additionally simplifies the definition of the safe state predicate, as we will see below.

Moreover, the experiment $\mathsf{Exp}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ also includes four predicates. Two of them are newly defined:

- $\mathsf{safe}_{\mathtt{P}}^{\mathsf{preK}}(\mathsf{ind})$: indicating whether ind's pre-key of party $\mathtt{P}$ is leaked or not. We define it as checking whether ind is included in the list $\mathcal{L}_{\mathtt{P}}^{\mathsf{rev}}$.
- $\mathsf{safe\text{-}st}_{\mathtt{P}}(t)$: indicating whether the state of party $\mathtt{P}$ at epoch t is expected to be safe or not. This predicate simplifies the definition of $\mathsf{safe\text{-}ch}_{\mathtt{P}}$ and $\mathsf{safe\text{-}inj}_{\mathtt{P}}$ predicates. We define it as checking whether any epoch from t to $(t - \triangle_{\mathsf{eSM}} + 1)$ is included in the list $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}}$.

The remaining two predicates were introduced in [1]. However, we define them in a different way in our model:

– $\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, t, \mathsf{ind})$: indicating whether the privacy of the message sent by $\mathtt{P}$ is expected to hold or not, under the randomness quality $\mathsf{flag} \in \{\mathsf{good}, \mathsf{bad}\}$, the sending epoch, and the receiver $\neg\mathtt{P}$'s ind. We define it to be true if and only if any of the following conditions hold:
  (a) both parties' states are safe at epoch t,
  (b) the partner $\neg\mathtt{P}$'s state is safe and the randomness quality is $\mathsf{flag} = \mathsf{good}$,
  (c) the partner $\neg\mathtt{P}$'s identity key is safe and the randomness quality is $\mathsf{flag} = \mathsf{good}$, or
  (d) the partner $\neg\mathtt{P}$'s pre-key is safe and the randomness quality is $\mathsf{flag} = \mathsf{good}$.
– $\mathsf{safe\text{-}inj}_\mathtt{P}(t)$: indicating whether the authenticity at the party $\mathtt{P}$'s epoch t (i.e., $\mathtt{P}$ is expected not to accept a forged ciphertext corresponding to epoch t) holds or not . We define it to be true if and only if the partner's state is safe at epoch t.

Compared to [1], our $\mathsf{safe\text{-}ch}_\mathtt{P}$ predicates additionally input a randomness quality, a epoch number, and a pre-key index. While the $\mathsf{safe\text{-}ch}_\mathtt{P}$ predicates in [1] equal the condition (a), our new conditions (b), (c), and (d) respectively capture the strong privacy, state compromise/failures, and PCR security properties. Moreover, our $\mathsf{safe\text{-}inj}_\mathtt{P}$ additionally inputs an epoch number t.

Our safe requirements are more relaxed and allow to reveal more information than in [1] (even when removing the usage of identity keys and pre-keys). In particular, if a safe predicate in the SM security model in [1] is true, then the one in our eSM model is true, but the reserve direction does not always hold.

Helping Functions. To simplify the security experiment definition, we use five helping functions. Four of them are introduced in [1], but we define some of them in our model with slight differences.

– **sam-if-nec**(r): If $r \neq \perp$, this function outputs (r, bad) indicating that the randomness is attacker-controlled. Otherwise, a new random string r is sampled from the space $\mathcal{R}$[1] and is output together with a flag good.
  This function is defined identically to the one in [1].
– **record**$(\mathtt{P}, \mathsf{type}, \mathsf{flag}, m, c)$: A record rec, which includes the party's identity $\mathtt{P}$, the partner's current pre-key counter $n_{\neg\mathtt{P}}$, the randomness flag flag, the epoch counter $t_\mathtt{P}$, the message index counter $i_\mathtt{P}$, the message m, and the ciphertext c, is added into the transcript sets trans and $\mathsf{allTrans}$. If the $\mathsf{safe\text{-}inj}_\mathtt{P}(t_\mathtt{P})$ predicate is false, then this record is also added into the compromise set comp. If c is a challenge ciphertext, indicated by whether $\mathsf{type} = \mathsf{chall}$, the record rec is also added into the challenge sets chall and $\mathsf{allChall}$.
  Compared to [1], the record in our function additionally includes $n_{\neg\mathtt{P}}$ and flag to simplify our identity key and pre-key reveal oracles. Moreover, the record is added to the compromise set only when the $\mathsf{safe\text{-}inj}_\mathtt{P}$ predicate is false, which means the partner's state is corrupted, capturing our strong authenticity.

[1] The randomness space $\mathcal{R}$ is not specific and depends on the concrete function where the output is expected to be used. Here, we use $\mathcal{R}$ only for simplicity.

- **ep-mgmt**$(\mathtt{P}, \mathsf{flag})$: When the party $\mathtt{P}$ enters a new epoch as the sender, the new epoch number is added to the state corruption list if the safe challenge predicate is false. Then, the epoch counter $t_{\mathtt{P}}$ is incremented by 1 and the message index counter i is initialized to 0.
  Due to the different definition of $\mathsf{safe\text{-}ch}_{\mathtt{P}}$ predicate, compared to [1], the condition in our **ep-mgmt** function additionally captures the impact of strong privacy on PCS.
- **delete**(t, i): deletes all records that includes (t, i) from the sets trans, chall, and comp.
  This function is identical to the one in [1] except for the syntax difference.

We also define a new helping function:

- **corruption-update**(): checks all records in the list whether the safe challenge predicates for the first messages in each epoch (still) hold or not. If it does not hold, then adds the epoch into the corruption list.

This helping function is used to capture the impact of the leakage of any secret on the safety of the state session. If the safe challenge predicate of the first message in an epoch is not true, then we mark the corresponding epoch as corrupted. Recall that this step is executed in **ep-mgmt** helping function whenever a sender enters a new epoch.

Experiment Execution and Oracles. The security experiment $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ includes eighteen oracles. Compared to the model in [1], our $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ security model additionally initializes the safe predicates for identity keys, the reveal and corruption lists for pre-keys and states, and the pre-key counters at the beginning of the experiment execution. Then, the attacker is given access to $\mathcal{O}_1 := \{\textsc{NewIdKey-A}, \textsc{NewIdKey-B}, \textsc{NewPreKey-A}, \textsc{NewPreKey-B}\}$ oracles for generating both parties' identity keys and at least one pre-keys. The rest of the experiment is similar to the one in [1]. A random initial shared secret iss is sampled from the space $\mathcal{ISS}$. Then, the session states $\mathsf{st}_{\mathtt{A}}$ and $\mathsf{st}_{\mathtt{B}}$ are respectively initialized by $\mathsf{eInit\text{-}A}$ and $\mathsf{eInit\text{-}B}$ of eSM. After initializing the epoch counters and message index counters, and the sets, the winning predicates $\mathsf{win}^{\mathsf{corr}}$ and $\mathsf{win}^{\mathsf{auth}}$, a challenge bit b is sampled uniformly at random. The attacker is given access to all oracles and terminates the experiment by outputting a bit b' for evaluating the winning condition $\mathsf{win}^{\mathsf{priv}}$. Finally, the experiment outputs all these three winning predicates. In Figure 1, we only depict the nine oracles with suffix $\text{-}A$ for party $\mathtt{A}$. The oracles for party $\mathtt{B}$ are defined analogously. The first eight oracles related to the identity keys and the pre-keys are new in our model.

- $\textsc{NewIdKey-A}(r)$, $\textsc{NewIdKey-B}(r)$: Each of these oracles can be queried at most once for each identity. The input random string, which is sampled when necessary, is used to produce a public-private identity key pair by using $\mathsf{IdKGen}(r)$. The corresponding safety flags are set according to whether the input $r = \perp$ or not. The public key is returned.
- $\textsc{NewPreKey-A}(r)$, $\textsc{NewPreKey-B}(r)$: Similar to the oracles above, a public-private pre-key pair is generated. The corresponding pre-key index is added into the list $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}$ or $\mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}$ if the input $r \neq \perp$. The public key is returned.

– RevIdKey-A, RevIdKey-B: These oracles simulate the reveal of the identity private key of a party P. The corresponding safe predicate is set to false. Then, the **corruption-update** helping function is invoked to update whether the session state safety predicate is entailed by this reveal. For each record in the all-challenge set allChall, we require that the safety predicate for any challenge ciphertext is not entailed. If the safe challenge predicate of any challenge record turns false, then this oracle undoes all actions during this invocation and exits. In particular, the corresponding safe predicate is reset to true. This step prevents the attacker from distinguishing the challenge bit by trivially revealing enough information to decrypt the challenge ciphertexts.
  Otherwise, this oracle checks whether the safe injection predicate of any record in the transcript set trans, which includes all ciphertexts that have not delivered, turns false. If so, then the corresponding record is added into the compromise set comp. This step prevents the attacker from making a trivial forgery by using the information leaked by the reveal of the identity key.
  Finally, the corresponding private identity key is returned.
– RevPreKey-A, RevPreKey-B: These oracles simulate the reveal of the current private pre-keys of a party P. Similar to above, we first adds the current pre-key index into the reveal list, followed by running **corruption-update** to check whether the session state leakage is entailed by this reveal. Next, for each record in the all-challenge set allChall, we require that the safety predicate for any challenge ciphertext is not entailed. If the safe challenge predicate of any challenge record turns false, then this oracle undoes all actions during this invocation and exits. In particular, the corresponding pre-key counter is removed from the reveal list.
  After that, this oracle checks whether the safe injection predicate of any record in the transcript set trans turns false. If so, then the corresponding record is added into the compromise set comp.
  Finally, the corresponding private pre-key is returned.

Compared to [1], the corruption oracles below are defined with huge differences:

– Corrupt-A, Corrupt-B: These oracles simulate the corruption of party P's session states. Notably, the session state does not include the private identity key and pre-key. First, the current epoch counter is added into the state corruption list, followed running **corruption-update** to update whether this corruption impacts the safety of other session states. Next, we require that either the chall does not include the record produced by the partner $\neg$P, which is identical to the requirement in the SM-security model in [1], or that either of the following two conditions holds: (1) the flag in the record is good and P's identity key is safe, or (2) the flag in the record is good and P's pre-key corresponding to the pre-key index in the record is safe. If the requirement is not satisfied, then this oracle undoes all actions during this invocation and exits. In particular, the corresponding epoch counter is removed from the corruption list. This requirement prevents the attacker from distinguishing the challenge bit by trivially revealing enough information to decrypt the challenge ciphertexts.

After that, we add all records $\mathsf{rec} \in \mathsf{trans}$, which are produced by $\neg\mathtt{P}$ at an unsafe epoch t (but not all epochs as in [1]), into the compromise set comp. We also add all records $\mathsf{rec} \in \mathsf{trans}$, which are produced by $\mathtt{P}$ at current epoch if the partner's session at current epoch is not safe. This requirement prevents the attacker from trivially breaking the strong authenticity (and therefore winning via the $\mathsf{win}^{\mathsf{auth}}$ predicate) by corrupting the sender's state and forging the corresponding undelivered messages.

Finally, the current epoch is added to $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$ and the session states are returned.

Compared to [1], the corruption oracles in our model can be queried under weaker requirements, providing the attacker with more information. Moreover, our corruption oracles set fewer records into the compromise set, which enables the attacker to forge ciphertexts corresponding to more epochs.

The remaining eight oracles are essentially identical to the ones in [1], except for syntax differences. For instance, sending message makes use of our eSend algorithm of eSM but not the Send algorithm of SM in [1] and the challenge predicate additionally inputs a flag, as we have already explained.

- Transmit-A(m, r), Transmit-B(m, r): These oracles simulate the real sending execution. The random string r is sampled when necessary. The epoch information is updated if entering a new epoch. After incrementing the message index, the eSend algorithm is executed using the controlled or freshly sampled randomness r to transmit the message m. After recording the transcript, the ciphertext is returned.
- Challenge-A(m_0, m_1, r), Challenge-B(m_0, m_1, r): These oracles simulate the sending execution, where the attacker tries to distinguish the encrypted message m_0 or m_1. These oracles are defined similar to the Transmit-A oracles. The only difference is that the safety predicate $\mathsf{safe\text{-}ch}_{\mathsf{P}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}})$ for $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ must hold and that the input messages m_0 and m_1 must have the same length. Moreover, the message to be sent by eSend is m_{b} where b is the challenge bit sampled at the beginning of the experiment.
- Deliver-A(ind, c), Deliver-B(ind, c): These oracles simulate the receiving execution of a ciphertext generated by the honest party. This means, there must exists a record $(\mathsf{P}, \mathsf{ind}, t, i, m, c)$ in the transcript set trans. The eRcv is invoked. If the output epoch t', message index i, and decrypted message m' does not match the one in the record, the attacker immediately wins via the predicate $\mathsf{win}^{\mathsf{corr}}$. If the output is in the challenge set chall, the decrypted message m' is set to $\perp$ to prevent the attacker from trivially distinguishing the challenge bit. After updating the epoch counter, the record is deleted from transcript set, challenge set, and compromise set. This in particular means that after accepting c, the ciphertext c is considered as a forgery. Finally, the output epoch t', the message index i, and the decrypted message m' is are returned.
- Inject-A(ind, c), Inject-B(ind, c): These oracles simulate a party $\mathtt{P}$'s receiving execution of a ciphertext forged by the attacker. These oracles are defined similar to the Deliver-A and Deliver-B oracles but with one more input $\mathsf{ind} \leq n_{\mathtt{P}}$, which specifies a pre-key for running eRcv. We require that eRcv

is invoked under the condition that c is not generated by the partner in the transcript set and that the safety predicate $\mathsf{safe\text{-}inj}_{\mathtt{P}}(t_{\mathtt{A}})$ and $\mathsf{safe\text{-}inj}_{\mathtt{P}}(t_{\mathtt{B}})$ both are true. Note that the safe injection predicates in the SM-security model in [1] are true only when both parties' session states are safe, while ours are true when the sender's session state is safe. Our requirement is relaxed than the one in [1].
If the decrypted message is not $\perp$ and the ciphertext at the same position is not compromised, the attacker immediately wins via the $\mathsf{win}^{\mathsf{auth}}$ predicate.
The rest of this oracle is identical to the one in the deliver oracles.

Even without taking the usage of identity keys and pre-keys into account, we our security model is strictly stronger than the one in [1].

Definition 3. *An* eSM *scheme Π is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-*eSM *secure if the below defined advantage for all attacker in time t is bounded by*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eSM}}_{\mathsf{eSM},\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \max\Big(&\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)],\\ &\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)],\\ &|\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|\Big) \leq \epsilon\end{aligned}$$

, where q, q_{ep}, and q_{M} respectively denote the maximal number of queries $\mathcal{A}$ can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ in Figure 1.

In particular, Definition 3 shows that no attacker against a eSM-secure scheme eSM can trigger the winning predicate $\mathsf{win}^{\mathsf{corr}}$ or $\mathsf{win}^{\mathsf{auth}}$ into true, or distinguish the challenge bit in the $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment in Figure 1 with respect to parameter $\triangle_{\mathsf{eSM}}$.

4.4 eSM Security and its Implication

Finally, we explain how our eSM security captures all security properties listed in Section 4.2.

- **Correctness:** No correctness means the encrypted message cannot be recovered correctly and causes the attacker to win via Line 61, even in the normal case where the attacker queries TRANSMIT-A or TRANSMIT-B and sends the output to the DELIVER-B or DELIVER-A oracles in the same order.
- **Immediate decryption and message-loss resilience (MLR):** The attacker can query TRANSMIT-A or TRANSMIT-B and send arbitrarily many output ciphertexts to DELIVER-B or DELIVER-A oracles in arbitrary order. No immediate decryption or message-loss resilience means the some of the messages cannot be recovered to the correct position from the delivered ciphertext, which causes the attacker to win via Line 61.
- **Forward secrecy (FS):** Note that all delivered messages are removed from the challenge set chall in Line 89. If all challenge ciphertexts have been delivered, then the attacker can freely access the corruption oracles.

```
$\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A})$:
    $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}, \mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{B}} \leftarrow \bot$
    $(n_{\mathtt{A}}, n_{\mathtt{B}}) \leftarrow (0,0)$
    $() \leftarrow \mathcal{A}^{\mathcal{O}_1}()$
    $\mathbf{req}\ \bot \notin \{\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}, \mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}\}$
    $\mathbf{req}\ n_{\mathtt{A}}, n_{\mathtt{B}} \geq 1$
    $iss \xleftarrow{\$} \mathcal{ISS}$
    $\mathsf{st}_{\mathtt{A}} \xleftarrow{\$} \mathsf{eInit\text{-}A}(iss),\ \mathsf{st}_{\mathtt{B}} \xleftarrow{\$} \mathsf{eInit\text{-}B}(iss)$
    $(t_{\mathtt{A}}, t_{\mathtt{B}}), (i_{\mathtt{A}}, i_{\mathtt{B}}) \leftarrow (0,0)$
    $\mathsf{trans}, \mathsf{chall}, \mathsf{comp}, \mathsf{allChall}, \mathsf{allTrans} \leftarrow \emptyset$
    $\mathsf{b} \xleftarrow{\$} \{0,1\},\ \mathsf{win}^{\mathsf{corr}}, \mathsf{win}^{\mathsf{auth}} \leftarrow \mathrm{false}$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_2}()$
    $\mathsf{win}^{\mathsf{priv}} \leftarrow [\![\mathsf{b} = \mathsf{b}']\!]$
    $\mathbf{return}\ (\mathsf{win}^{\mathsf{corr}}, \mathsf{win}^{\mathsf{auth}}, \mathsf{win}^{\mathsf{priv}})$

$\textsc{NewIdKey-A}(r)$:
    $\mathbf{req}\ \mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}} = \bot$
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathbf{sam\text{-}if\text{-}nec}(r)$
    $(ipk_{\mathtt{A}}, ik_{\mathtt{A}}) \xleftarrow{\$} \mathsf{IdKGen}(r)$
    $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}} \leftarrow [\![\mathsf{flag} = \mathsf{good}]\!]$
    $\mathbf{return}\ ipk_{\mathtt{A}}$

$\textsc{NewPreKey-A}(r)$:
    $n_{\mathtt{A}}{+}{+}$
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathbf{sam\text{-}if\text{-}nec}(r)$
    $(prepk^{n_{\mathtt{A}}}_{\mathtt{A}}, prek^{n_{\mathtt{A}}}_{\mathtt{A}}) \xleftarrow{\$} \mathsf{PreKGen}(r)$
    $\mathbf{if}\ \mathsf{flag} = \mathsf{bad} : \mathcal{L}^{\mathsf{rev}}_{\mathtt{A}} \xleftarrow{+} n_{\mathtt{A}}$
    $\mathbf{return}\ prepk_{\mathtt{A}}$

$\textsc{RevIdKey-A}$:
    $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}} \leftarrow \mathrm{false}$
    $\mathbf{corruption\text{-}update}()$
    $\mathbf{foreach}\ (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c) \in \mathsf{allChall}$
        $\mathbf{req}\ \mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, t, \mathsf{ind})$
    $\mathbf{foreach}\ (\mathtt{P}, t) \in \mathsf{trans}\ \mathbf{and}\ \neg\mathsf{safe\text{-}inj}_{\neg\mathtt{P}}(t)$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathtt{P}, t)$
    $\mathbf{return}\ ik_{\mathtt{A}}$

$\textsc{RevPreKey-A}$:
    $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}} \xleftarrow{+} n_{\mathtt{A}}$
    $\mathbf{corruption\text{-}update}()$
    $\mathbf{foreach}\ (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c) \in \mathsf{allChall}$
        $\mathbf{req}\ \mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, t, \mathsf{ind})$
    $\mathbf{foreach}\ (\mathtt{P}, t) \in \mathsf{trans}\ \mathbf{and}\ \neg\mathsf{safe\text{-}inj}_{\neg\mathtt{P}}(t)$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathtt{P}, t)$
    $\mathbf{return}\ prek^{n_{\mathtt{A}}}_{\mathtt{A}}$

$\textsc{Corrupt-A}$:
    $\mathcal{L}^{\mathsf{cor}}_{\mathtt{A}} \xleftarrow{+} t_{\mathtt{A}}$
    $\mathbf{corruption\text{-}update}()$
    $\mathbf{req}\ (\mathtt{B}, \mathsf{ind}, \mathsf{flag}) \notin \mathsf{chall}\ \mathbf{or}\ \big(\mathsf{flag} = \mathsf{good}\ \mathbf{and}\ \mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}\big)\ \mathbf{or}\ \big(\mathsf{flag} = \mathsf{good}\ \mathbf{and}\ \mathsf{safe}^{\mathsf{preK}}_{\mathtt{A}}(\mathsf{ind})\big)$
    $\mathbf{foreach}\ (\mathtt{B}, t) \in \mathsf{trans}\ \mathbf{and}\ \neg\mathsf{safe\text{-}st}_{\mathtt{B}}(t)$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathtt{B}, t)$
    $\mathbf{foreach}\ (\mathtt{A}, t_{\mathtt{A}}) \in \mathsf{trans}\ \mathbf{and}\ \neg\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}})$
        $\mathsf{comp} \xleftarrow{+} \mathsf{trans}(\mathtt{A}, t_{\mathtt{A}})$
    $\mathbf{return}\ \mathsf{st}_{\mathtt{A}}$

$\textsc{Transmit-A}(m, r)$:
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathbf{sam\text{-}if\text{-}nec}(r)$
    $\mathbf{ep\text{-}mgmt}(\mathtt{A}, \mathsf{flag})$
    $i_{\mathtt{A}}{+}{+}$
    $(\mathsf{st}_{\mathtt{A}}, c) \xleftarrow{\$} \mathsf{eSend}(\mathsf{st}_{\mathtt{A}}, ipk_{\mathtt{B}}, prepk^{n_{\mathtt{B}}}_{\mathtt{B}}, m; r)$
    $\mathbf{record}(\mathtt{A}, \mathsf{norm}, \mathsf{flag}, m, c)$
    $\mathbf{return}\ c$

$\textsc{Challenge-A}(m_0, m_1, r)$:
    $(r, \mathsf{flag}) \xleftarrow{\$} \mathbf{sam\text{-}if\text{-}nec}(r)$
    $\mathbf{ep\text{-}mgmt}(\mathtt{A}, \mathsf{flag})$
    $\mathbf{req}\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}})\ \mathbf{and}\ |m_0| = |m_1|$
    $i_{\mathtt{A}}{+}{+}$
    $(\mathsf{st}_{\mathtt{A}}, c) \xleftarrow{\$} \mathsf{eSend}(\mathsf{st}_{\mathtt{A}}, ipk_{\mathtt{B}}, prepk^{n_{\mathtt{B}}}_{\mathtt{B}}, m_{\mathsf{b}}; r)$
    $\mathbf{record}(\mathtt{A}, \mathsf{chall}, \mathsf{flag}, m_{\mathsf{b}}, c)$
    $\mathbf{return}\ c$

$\textsc{Deliver-A}(c)$:
    $\mathbf{req}\ (\mathtt{B}, \mathsf{ind}, t, i, m, c) \in \mathsf{trans}$ for some $\mathsf{ind}, t, i, m$
    $(\mathsf{st}_{\mathtt{A}}, t', i', m') \leftarrow \mathsf{eRcv}(\mathsf{st}_{\mathtt{A}}, ik_{\mathtt{A}}, prek^{\mathsf{ind}}_{\mathtt{A}}, c)$
    $\mathbf{if}\ (t', i', m') \neq (t, i, m)$: $\mathsf{win}^{\mathsf{corr}} \leftarrow \mathrm{true}$
    $\mathbf{if}\ (t, i, m) \in \mathsf{chall}$: $m' \leftarrow \bot$
    $t_{\mathtt{A}} \leftarrow \max(t_{\mathtt{A}}, t')$
    $\mathbf{delete}(t, i)$
    $\mathbf{return}\ (t', i', m')$

$\textsc{Inject-A}(\mathsf{ind}, c)$:
    $\mathbf{req}\ (\mathtt{B}, c) \notin \mathsf{trans}\ \mathbf{and} \mathsf{ind} \leq n_{\mathtt{A}}$
    $\mathbf{req}\ \mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{B}})\ \mathbf{and}\ \mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{A}})$
    $(\mathsf{st}_{\mathtt{A}}, t', i', m') \leftarrow \mathsf{eRcv}(\mathsf{st}_{\mathtt{A}}, ik_{\mathtt{A}}, prek^{\mathsf{ind}}_{\mathtt{A}}, c)$
    $\mathbf{if}\ m' \neq \bot\ \mathbf{and}\ (\mathtt{B}, t', i') \notin \mathsf{comp}$
        $\mathsf{win}^{\mathsf{auth}} \leftarrow \mathrm{true}$
    $t_{\mathtt{A}} \leftarrow \max(t_{\mathtt{A}}, t')$
    $\mathbf{delete}(t', i')$
    $\mathbf{return}\ (t', i', m')$
```

Fig. 1: The extended secure messaging experiment $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$. $\mathcal{O}_1 := \{\textsc{NewIdKey-A}, \textsc{NewIdKey-B}, \textsc{NewPreKey-A}, \textsc{NewPreKey-B}\}$ and $\mathcal{O}_2$ denotes all oracles. This figure only depicts the oracles for $\mathtt{A}$ (ending with -A). The oracles for $\mathtt{B}$ are defined analogously. We highlighted the difference to the SM-security game for a SM scheme in [1] with blue color. We give more helping functions and safe predicates in Figure 2.

```
$\textbf{sam-if-nec}(r)$:
74  $\mathsf{flag} \leftarrow \mathsf{bad}$
75  if $r = \perp$
76      $r \xleftarrow{\$} \mathcal{R}$
77      $\mathsf{flag} \leftarrow \mathsf{good}$
78  return $(r, \mathsf{flag})$
$\textbf{record}(\mathtt{P}, \mathsf{type}, \mathsf{flag}, m, c)$:
79  $\mathsf{rec} \leftarrow (\mathtt{P}, n_{\neg\mathtt{P}}, \mathsf{flag}, t_\mathtt{P}, i_\mathtt{P}, m, c)$
80  $\mathsf{allTrans}, \mathsf{trans} \xleftarrow{+} \mathsf{rec}$
81  if $\neg\mathsf{safe\text{-}inj}_{\neg\mathtt{P}}(t_\mathtt{P})$: $\mathsf{comp} \xleftarrow{+} \mathsf{rec}$
82  if $\mathsf{type} = \mathsf{chall}$: $\mathsf{allChall}, \mathsf{chall} \xleftarrow{+} \mathsf{rec}$
$\textbf{ep-mgmt}(\mathtt{P}, \mathsf{flag})$:
83  if $(\mathtt{P} = \mathtt{A}$ and $t_\mathtt{P}$ even$)$ or $(\mathtt{P} = \mathtt{B}$ and $t_\mathtt{P}$ odd$)$
84      if $\neg\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, t_\mathtt{P}, n_{\neg\mathtt{P}})$
85          $\mathcal{L}^{\mathsf{cor}}_\mathtt{P} \xleftarrow{+} t_\mathtt{P} + 1$
86      $t_\mathtt{P}{+}{+}$
87      $i_\mathtt{P} \leftarrow 0$
$\textbf{delete}(t, i)$:
88  $\mathsf{rec} \leftarrow (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c)$ for some $\mathtt{P}, \mathsf{ind}, \mathsf{flag}, m, c$
89  $\mathsf{trans}, \mathsf{chall}, \mathsf{comp} \xleftarrow{-} \mathsf{rec}$
$\textbf{corruption-update}()$:
90  foreach $(\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, 1, m, c) \in \mathsf{allTrans}$
91      if $\neg\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, (t-1), \mathsf{ind})$
92          $\mathcal{L}^{\mathsf{cor}}_\mathtt{P} \xleftarrow{+} t$
```

$\mathsf{safe}^{\mathsf{preK}}_\mathtt{P}(\mathsf{ind}) :\Leftrightarrow \mathsf{ind} \notin \mathcal{L}^{\mathsf{rev}}_\mathtt{P}$

$\mathsf{safe\text{-}st}_\mathtt{P}(t) :\Leftrightarrow t, (t-1), \ldots, (t - \triangle_{\mathsf{eSM}} + 1) \notin \mathcal{L}^{\mathsf{cor}}_\mathtt{P}$

$\mathsf{safe\text{-}ch}_\mathtt{P}(\mathsf{flag}, t, \mathsf{ind}) :\Leftrightarrow \big(\mathsf{safe\text{-}st}_\mathtt{P}(t) \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(t)\big) \textbf{ or } \big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(t)\big) \textbf{ or } \big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{idK}}_{\neg\mathtt{P}}\big) \textbf{ or } \big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{preK}}_{\neg\mathtt{P}}(\mathsf{ind})\big)$

$\mathsf{safe\text{-}inj}_\mathtt{P}(t) :\Leftrightarrow \mathsf{safe\text{-}st}_{\neg\mathtt{P}}(t)$

Fig. 2: The helping functions in extended secure messaging experiment $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$. We highlight the difference to the SM-security game for a SM scheme in [1] with blue color.

No FS means that the attacker can distinguish the challenge bit from the past encrypted messages, which means the attacker wins via Line 12.

– **Post-compromise security (PCS):** Note that the state leakage epoch will not be added into the state corruption list in Line 85, if the challenge predicate is true in Line 84 (assuming fresh randomness and the partner's uncompromised session state, or identity key or pre-key).
  No PCS means that a passive attacker can still corrupt the states even after the corrupted party sends a reply in a new epoch, which further causes the lose of other security properties.
– ***Strong* authenticity:** The attacker can inject a forged ciphertext (Line 66) that does not correspond to a compromised ciphertext position (Line 69) if sender's session state is safe. Recall that a ciphertext is compromised only when the session state of the sender is unsafe (see Line 28, 35, 41, 44, 81).
  No strong authenticity means that the forged ciphertext can be decrypted to a non-$\perp$ message when the sender is not corrupted, and further causes the winning of the attacker via Line 70.
– ***Strong* privacy**: Note that the challenge ciphertexts must be produced without the violation the safety predicate safe-ch in Line 54, which means at least one of the following combinations are not leaked: (1) both parties' states, (2) the encryption randomness and the receiver's state, (3) the encryption

randomness and the receiver's private identity key, or (4) the encryption randomness and the receiver's corresponding private pre-key. Moreover, our identity key reveal oracles (RevIdKey-A, RevIdKey-B), pre-key reveal oracles (RevPreKey-A and RevPreKey-B) oracles, and state corruption oracles (Corrupt-A and Corrupt-B) also prevent the attacker from knowing all of the above combinations related to any challenge ciphertext at the same time (see Line 27, 34, 40).
No strong privacy means that the attacker can distinguish the challenge bit even when at least one of the above four combinations holds, which further causes the winning of the attacker win via Line 12.

– **Randomness leakage/failures:** This is ensured by the fact that all of the above properties hold if the parties' session states are uncompromised.
– ***State compromise/failures*****:** This is ensured by the strong privacy even when both parties' state are corrupted, as explained above.
– ***Periodic confidentiality recovery*** **(PCR):** Note that the pre-keys can be periodically generated optionally under fresh randomness. The PCR is ensured by the strong privacy when the sender's randomness is good and the receiver's newly freshly sampled pre-key is safe, as explained above.

Moreover, we can also observe that higher security can be obtained if the device of a party (assume A) supports a secure environment, such as an HSM. If A's identity keys are generated in a more secure environment, the randomness during the generation can be neither manipulated nor predicted by any attacker. This means that the attacker can only query NewIdKey-A(r) with input $r = \perp$ in the $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. Furthermore, since A's private identity key is stored in the secure environment, the attacker cannot reveal this key. This means that the attacker can never query RevIdKey-A oracle in $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, the predicate $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}$ is always true. If the partner B has access to the fresh randomness, then the privacy of the messages sent from B to A always holds.

5 Extended Secure Messaging Scheme

In this section, we present our eSM construction in Section 5.1. In Section 5.2, we show that our eSM construction provides eSM security. Finally in Section 5.3, we compare our protocol to known secure messaging protocols with immediate decryption: ACD19 and ACD19-PK [1].

5.1 The eSM Construction

Our eSM construction is depicted in Figure 3. For simplicity, we assume all symmetric keys in our construction (including the root key rk, the chain key ck, the unidirectional ratchet key urk, and the message key mk) have the same domain $\{0,1\}^\lambda$. We assume the key generation randomness spaces of KEM and DS are also $\{0,1\}^\lambda$. Note that each eSM scheme involves the states of each party. We start with the definition of the state in our eSM construction.

```
IdKGen():
  $(ipk, ik) \xleftarrow{\$} \mathsf{K.KG}()$
  $\mathbf{return}\ (ipk, ik)$

PreKGen():
  $(prepk, prek) \xleftarrow{\$} \mathsf{K.KG}()$
  $\mathbf{return}\ (prepk, prek)$

eInit-A($iss$):
  $\mathsf{st}_{\mathtt{A}}.rk \parallel \mathsf{st}_{\mathtt{A}}.ck^0 \parallel r_{\mathtt{A}}^{\mathsf{KEM}} \parallel r_{\mathtt{B}}^{\mathsf{KEM}} \parallel r_{\mathtt{A}}^{\mathsf{DS}} \parallel r_{\mathtt{B}}^{\mathsf{DS}} \leftarrow iss$ , $\mathsf{st}_{\mathtt{A}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$
  $(\_, \mathsf{st}_{\mathtt{A}}.\mathsf{dk}^0) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{A}}^{\mathsf{KEM}})$, $(\mathsf{st}_{\mathtt{A}}.\mathsf{ek}^1, \_) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{B}}^{\mathsf{KEM}})$
  $(\mathsf{st}_{\mathtt{A}}.\mathsf{sk}^{-1}, \_) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{A}}^{\mathsf{DS}})$, $(\_, \mathsf{st}_{\mathtt{A}}.\mathsf{vk}^0) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{B}}^{\mathsf{DS}})$
  $\mathsf{st}_{\mathtt{A}}.\mathsf{id} \leftarrow \mathtt{A}$, $\mathsf{st}_{\mathtt{A}}.\mathsf{prtr} \leftarrow \perp$, $\mathsf{st}_{\mathtt{A}}.t \leftarrow 0$, $\mathsf{st}_{\mathtt{A}}.i \leftarrow 0$, $\mathsf{st}_{\mathtt{A}}.\mathcal{D}_l[\cdot] \leftarrow \perp$, $\mathsf{st}_{\mathtt{A}}.\mathcal{D}_{urk}^0[\cdot] \leftarrow \perp$
  $\mathbf{return}\ \mathsf{st}_{\mathtt{A}}$

eInit-B($iss$):
  $\mathsf{st}_{\mathtt{B}}.rk \parallel \mathsf{st}_{\mathtt{B}}.ck^0 \parallel r_{\mathtt{A}}^{\mathsf{KEM}} \parallel r_{\mathtt{B}}^{\mathsf{KEM}} \parallel r_{\mathtt{A}}^{\mathsf{DS}} \parallel r_{\mathtt{B}}^{\mathsf{DS}} \leftarrow iss$ , $\mathsf{st}_{\mathtt{B}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$
  $(\mathsf{st}_{\mathtt{B}}.\mathsf{ek}^0, \_) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{A}}^{\mathsf{KEM}})$, $(\_, \mathsf{st}_{\mathtt{B}}.\mathsf{dk}^1) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{B}}^{\mathsf{KEM}})$
  $(\_, \mathsf{st}_{\mathtt{B}}.\mathsf{vk}^{-1}) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{A}}^{\mathsf{DS}})$, $(\mathsf{st}_{\mathtt{B}}.\mathsf{sk}^0, \_) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{B}}^{\mathsf{DS}})$
  $\mathsf{st}_{\mathtt{B}}.\mathsf{id} \leftarrow \mathtt{B}$, $\mathsf{st}_{\mathtt{B}}.\mathsf{prtr} \leftarrow \perp$, $\mathsf{st}_{\mathtt{B}}.t \leftarrow 0$, $\mathsf{st}_{\mathtt{B}}.i \leftarrow 0$, $\mathsf{st}_{\mathtt{B}}.\mathcal{D}_l[\cdot] \leftarrow \perp$
  $\mathbf{return}\ \mathsf{st}_{\mathtt{B}}$

eSend(st, $ipk, prepk, m$):
  $(c_1, k_1) \xleftarrow{\$} \mathsf{K.Enc}(\mathsf{st.ek}^{\mathsf{st}.t})$, $(c_2, k_2) \xleftarrow{\$} \mathsf{K.Enc}(ipk)$, $(c_3, k_3) \xleftarrow{\$} \mathsf{K.Enc}(prepk)$
  $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$
  $\mathbf{if}$ $(\mathsf{st.id} = \mathtt{A}$ $\mathbf{and}$ $\mathsf{st}.t\ even)$ $\mathbf{or}$ $(\mathsf{st.id} = \mathtt{B}$ $\mathbf{and}$ $\mathsf{st}.t\ odd)$
    $l \leftarrow \mathsf{eSend\text{-}Stop(st)}$, $\mathsf{st}.t{+}{+}$, $\mathsf{st}.i \leftarrow 0$
    $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}.nxs, r)$
    $(\mathsf{ek}, \mathsf{st.dk}^{\mathsf{st}.t+1}) \xleftarrow{\$} \mathsf{K.KG}(r^{\mathsf{KEM}})$, $(\mathsf{st.sk}^{\mathsf{st}.t}, \mathsf{vk}) \xleftarrow{\$} \mathsf{D.KG}(r^{\mathsf{DS}})$
    $\mathsf{st.prtr} \leftarrow (l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk})$, $\sigma^{\mathsf{ar}} \leftarrow \mathsf{D.Sign}(\mathsf{st.sk}^{\mathsf{st}.t-2}, \mathsf{st.prtr})$
    $\mathsf{st.prtr} \leftarrow (\mathsf{st.prtr}, \sigma^{\mathsf{ar}})$, $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$
  $(\mathsf{st}.ck^{\mathsf{st}.t}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{\mathsf{st}.t})$, $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})$, $c' \leftarrow \mathsf{S.Enc}(mk, m)$
  $\mathsf{prtr}^{\mathsf{ur}} \leftarrow (\mathsf{st}.t, \mathsf{st}.i, c', c_1, c_2, c_3)$, $\sigma^{\mathsf{ur}} \leftarrow \mathsf{D.Sign}(\mathsf{st.sk}^{\mathsf{st}.t}, (\mathsf{st.prtr}, \mathsf{prtr}^{\mathsf{ur}}))$
  $\mathbf{return}\ (\mathsf{st}, (\mathsf{st.prtr}, \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}}))$

eRcv(st, $ik, prek, c$):
  $(\mathsf{prtr}^{\mathsf{ar}}, \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}}) \leftarrow c$, $((l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk}), \sigma^{\mathsf{ar}}) \leftarrow \mathsf{prtr}^{\mathsf{ar}}$ ,$(t, i, c', c_1', c_2', c_3') \leftarrow \mathsf{prtr}^{\mathsf{ur}}$
  $\mathbf{if}$ $t \leq \mathsf{st}.t - 2$: $\mathbf{req}$ $\mathsf{st}.\mathcal{D}_l[t] \neq \perp$ $\mathbf{and}$ $i \leq \mathsf{st}.\mathcal{D}_l[t]$
  $\mathbf{req}$ $t \leq \mathsf{st}.t + 1$ $\mathbf{and}$ $\Big((\mathsf{st.id} = \mathtt{A}$ $\mathbf{and}$ $t\ even)$ $\mathbf{or}$ $(\mathsf{st.id} = \mathtt{B}$ $\mathbf{and}$ $t\ odd)\Big)$
  $\mathbf{if}$ $t = \mathsf{st}.t + 1$
    $\mathbf{req}$ $\mathsf{D.Vrfy}(\mathsf{st.vk}^{t-2}, (l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk}), \sigma^{\mathsf{ar}})$
    $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$, $\mathsf{st}.\mathcal{D}_l[t-2] \leftarrow l$, $\mathsf{st}.t{+}{+}$
    $k_1 \leftarrow \mathsf{K.Dec}(\mathsf{st.dk}^{\mathsf{st}.t}, c_1)$, $k_2 \leftarrow \mathsf{K.Dec}(ik, c_2)$, $k_3 \leftarrow \mathsf{K.Dec}(prek, c_3)$
    $(\mathsf{upd}^{\mathsf{ar}}, \_) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$, $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$
    $\mathcal{D}_{urk}^{\mathsf{st}.t}[\cdot] \leftarrow \perp$, $\mathsf{st}.i \leftarrow 0$, $\mathsf{st.ek}^{\mathsf{st}.t+1} \leftarrow \mathsf{ek}$, $\mathsf{st.vk}^{\mathsf{st}.t} \leftarrow \mathsf{vk}$
  $\mathbf{req}$ $\mathsf{D.Vrfy}(\mathsf{st.vk}^t, (\mathsf{prtr}^{\mathsf{ar}}, \mathsf{prtr}^{\mathsf{ur}}), \sigma^{\mathsf{ur}})$
  $k_1' \leftarrow \mathsf{K.Dec}(\mathsf{st.dk}^{\mathsf{st}.t}, c_1')$, $k_2' \leftarrow \mathsf{K.Dec}(ik, c_2')$, $k_3' \leftarrow \mathsf{K.Dec}(prek, c_3')$
  $(\_, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1', k_2', k_3')$
  $\mathbf{while}$ $\mathsf{st}.i \leq i$: $(\mathsf{st}.ck^{\mathsf{st}.t}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{\mathsf{st}.t})$, $\mathcal{D}_{urk}^{\mathsf{st}.t}[\mathsf{st}.i] \leftarrow urk$, $\mathsf{st}.i{+}{+}$
  $urk \leftarrow \mathcal{D}_{urk}^{\mathsf{st}.t}[i]$, $\mathcal{D}_{urk}^{\mathsf{st}.t}[i] \leftarrow \perp$, $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})$, $m \leftarrow \mathsf{S.Dec}(mk, c')$
  $\mathbf{return}\ (\mathsf{st}, t, i, m)$
```

Fig. 3: Our eSM construction. The KEM denotes a key encapsulation mechanism scheme. The DS denotes a digital signature scheme. The SKE denotes an authenticated encryption scheme. The KDF_i for $i \in [5]$ denote five independent key derivation functions.

Definition 4. *The state in our* eSM *construction in Figure 3 consists of following variables:*

- $\mathsf{st.id}$*: the owner of state. In this paper, we have* $\mathsf{st_A.id} = \mathtt{A}$ *and* $\mathsf{st_B.id} = \mathtt{B}$.
- $\mathsf{st}.t$*: the local epoch counter. It starts with* 0.
- $\mathsf{st}.i$*: the local message index counter. It starts with* 0.
- $\mathsf{st}.rk \in \{0,1\}^\lambda$*: the (symmetric) root key. This key is updated only when entering next epoch and is used to initialize the chain key. This key is initialized from the initial shared secret.*
- $\mathsf{st}.ck^0, \mathsf{st}.ck^1, ... \in \{0,1\}^\lambda$*: the (symmetric) chain key at each epoch. This key is initialized at the beginning of each epoch and is used to derive unidirectional ratchet key (urk).*
- $\mathsf{st}.nxs \in \{0,1\}^\lambda$*: a local NAXOS random string, which is used to improve the randomness when generating new* KEM *and* DS *key pairs.*
- $\mathsf{st}.\mathcal{D}_l$*: the dictionary that stores the maximal number (we also say the length) of the transmissions in the previous epochs.*
- $\mathsf{st.prtr}$*: the pre-transcript produced by the sender for updating the root key. This value is generated (and locally stored) at the beginning of each epoch and is attached to the ciphertext whenever sending messages in the same epoch.*
- $\mathsf{st}.\mathcal{D}^0_{urk}, \mathsf{st}.\mathcal{D}^1_{urk}, ...$*: the dictionaries that store the unidirectional ratchet keys urk for each epoch. The urk is used to derive the message key for encrypting each message or decrypting each ciphertext using* SKE.
- $\mathsf{st.ek}^0, \mathsf{st.ek}^1, ...$*: the (asymmetric)* KEM *public keys. These keys are used to encapsulate the randomness, which (together with the unidirectional ratchet key) is used to derive the message keys.*
- $\mathsf{st.dk}^0, \mathsf{st.dk}^1, ...$*: the (asymmetric)* KEM *private key.s These keys are used to decapsulate the randomness, which (together with the unidirectional ratchet key) is used to derive the message keys.*
- $\mathsf{st.sk}^{-1}, \mathsf{st.sk}^0, \mathsf{st.sk}^1, ...$*: the (asymmetric)* DS *private keys. These keys are used to sign the (other) pre-transcript output by* eSend[2].
- $\mathsf{st.vk}^{-1}, \mathsf{st.vk}^0, \mathsf{st.vk}^1, ...$*: the (asymmetric)* DS *public keys. These keys are used to verify the signatures, which is signed by the corresponding* DS *private key.*

Our eSM construction makes use of two auxiliary functions: eSend-Stop and eRcv-Max for practical memory management. Similar to [1], we only explain the underlying mechanism and omit their concrete instantiation.

- eSend-Stop(st): This algorithm is called in eSend algorithm when the caller switches its role from the message receiver in epoch $\mathsf{st}.t$ to the message sender in a new epoch $\mathsf{st}.t + 1$. This algorithm inputs (the caller's) state st and outputs how many messages are sent in the old epoch (i.e., $\mathsf{st}.t - 1$, when the state owner acts as the sender), denoted by l. During the algorithm, the state values for sending messages in the old epochs, i.e., $\mathsf{st}.ck^t$ and $\mathsf{st.ek}^t$ such that

[2] The superscript of the signing keys verification keys are epochs when the DS key pairs are generated and used until the next key generation two epochs later. Here, we slightly abuse the notation and have $\mathsf{st.sk}^{-1}$ and $\mathsf{st.vk}^{-1}$, which are used only to sign and verify the next verification key in epoch 1.

$t = \mathsf{st}.t - 1$, are erased. The signing key $\mathsf{st}.\mathsf{sk}^t$ is also erased after its signs the next verification key $\mathsf{st}.\mathsf{vk}^{t+2}$ later. We write $l \leftarrow \mathsf{eSend\text{-}Stop}(\mathsf{st})$.

– $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$: This algorithm is called in eRcv algorithm when the caller switches its role from message sender in epoch $\mathsf{st}.t$ to the message receiver in a new epoch $\mathsf{st}.t + 1$. This algorithm inputs (the caller's) state st and a number l and remembers the value l together with the epoch counter $t' := \mathsf{st}.t - 1$ locally. As long as l messages corresponds to the old epoch t' are received, the state values for receiving messages in epoch t', i.e., $\mathsf{st}.ck^{t'}$, $\mathsf{st}.\mathsf{dk}^{t'}$, $\mathsf{st}.\mathsf{vk}^{t'}$, $\mathsf{st}.\mathcal{D}_{urk}^{t'}$, $\mathsf{st}.\mathcal{D}_l[t']$ are erased, i.e., set to $\perp$.

Following the syntax in Definition 1, our eSM construction consists of six algorithms, each of which is explained in details below.

$\mathsf{IdKGen}()$*:* The identity key generation algorithm samples and outputs a public-private KEM key pair.

$\mathsf{PreKGen}()$*:* The pre-key generation algorithm samples and outputs a public-private KEM key pair.

$\mathsf{eInit\text{-}A}(iss)$*:* The $\mathtt{A}$'s extended initialization algorithm inputs an initial shared secret $iss \in \mathcal{ISS}$. First, $\mathtt{A}$ parses iss into six components: the shared root key $\mathsf{st}_{\mathtt{A}}.rk$, the shared chain key $\mathsf{st}_{\mathtt{A}}.ck^0$, and four randomness for $\mathtt{A}$'s and $\mathtt{B}$'s KEM and DS key generation: $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, $r_{\mathtt{B}}^{\mathsf{DS}}$. Next, $\mathtt{A}$ samples a random string $\mathsf{st}_{\mathtt{A}}.nxs$. Then, $\mathtt{A}$ respectively runs $\mathsf{K.KG}$ and $\mathsf{D.KG}$ on the above randomness and stores $\mathsf{st}_{\mathtt{A}}.\mathsf{dk}^0$, $\mathsf{st}_{\mathtt{A}}.\mathsf{ek}^1$, $\mathsf{st}_{\mathtt{A}}.\mathsf{sk}^{-1}$, $\mathsf{st}_{\mathtt{A}}.\mathsf{vk}^0$, which are respectively generated using $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, and $r_{\mathtt{B}}^{\mathsf{DS}}$. The other values generated in the meantime are discarded.

Finally, $\mathtt{A}$ sets the identity $\mathsf{st}_{\mathtt{A}}.\mathsf{id}$ to $\mathtt{A}$, the local pre-transcript $\mathsf{st}_{\mathtt{A}}.\mathsf{prtr}$ to $\perp$, the epoch counter $\mathsf{st}_{\mathtt{A}}.t$ to 0, the message index $\mathsf{st}_{\mathtt{A}}.i$ to 0, and initializes the maximal transmission length dictionary $\mathcal{D}_l$ and the unidirectional ratchet dictionary $\mathcal{D}_{urk}^0$, followed by outputting the state $\mathsf{st}_{\mathtt{A}}$.

$\mathsf{eInit\text{-}B}(iss)$*:* The $\mathtt{B}$'s extended initialization algorithm inputs an initial shared secret $iss \in \mathcal{ISS}$ and runs very similar to $\mathsf{eInit\text{-}A}$. First, $\mathtt{B}$ parses iss into six components: the shared root key $\mathsf{st}_{\mathtt{B}}.rk$, the shared chain key $\mathsf{st}_{\mathtt{B}}.ck^0$, and four randomness for $\mathtt{A}$'s and $\mathtt{B}$'s KEM and DS key generation: $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, $r_{\mathtt{B}}^{\mathsf{DS}}$. Next, $\mathtt{B}$ samples a random string $\mathsf{st}_{\mathtt{B}}.nxs$. Then, $\mathtt{B}$ respectively runs $\mathsf{K.KG}$ and $\mathsf{D.KG}$ on the above randomness and stores $\mathsf{st}_{\mathtt{B}}.\mathsf{ek}^0$, $\mathsf{st}_{\mathtt{B}}.\mathsf{dk}^1$, $\mathsf{st}_{\mathtt{B}}.\mathsf{vk}^{-1}$, $\mathsf{st}_{\mathtt{A}}.\mathsf{sk}^0$, which are respectively generated using $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, and $r_{\mathtt{B}}^{\mathsf{DS}}$. The other values generated in the meantime are discarded. Note that the values stored by $\mathtt{B}$ is the ones discarded by $\mathtt{A}$, and vice versa.

Finally, $\mathtt{B}$ sets the identity $\mathsf{st}_{\mathtt{B}}.\mathsf{id}$ to $\mathtt{B}$, the local pre-transcript $\mathsf{st}_{\mathtt{B}}.\mathsf{prtr}$ to $\perp$, the epoch counter $\mathsf{st}_{\mathtt{B}}.t$ to 0, the message index $\mathsf{st}_{\mathtt{B}}.i$ to 0, and initializes the maximal transmission length dictionary $\mathcal{D}_l$, followed by outputting the state $\mathsf{st}_{\mathtt{B}}$. Note that no unidirectional ratchet dictionary $\mathcal{D}_{urk}^0$ is initialized, since $\mathtt{B}$ acts as the sender in the epoch 0.

$\mathsf{eSend}(\mathsf{st}, ipk, prepk, m)$*:* The sending algorithm inputs the (caller's) state st, the (caller's partner's) public identity key ipk and pre-key $prepk$, and a message m.

First, the caller runs the encapsulation algorithm of KEM and obtains three ciphertext-key tuples (c_1, k_1), (c_2, k_2), and (c_3, k_3) respectively using the local key $\mathsf{st.ek}^{\mathsf{st}.t}$, and the identity key ipk, and the pre-key $prepk$. Next, the caller applies KDF_1 to k_1, k_2, and k_3, for deriving two update values $\mathsf{upd}^{\mathsf{ar}}$ and $\mathsf{upd}^{\mathsf{ur}}$.

If the caller switches its role from receiver to sender, i.e. the caller $\mathsf{st.id}$ is $\mathtt{A}$ and the epoch $\mathsf{st}_{\mathtt{A}}.t$ is even or the caller is $\mathtt{B}$ and the epoch is odd, it first executes the following so-called *asymmetric ratchet* (ar) framework: First, the caller runs $\mathsf{eSend\text{-}Stop}(\mathsf{st})$ for a value l that counts the sent messages in the previous epoch, followed by incrementing the epoch counter $\mathsf{st}.t$ by 1 and initializing the message index counter i to 0. Next, the caller samples a random string r, which together with the local NAXOS string $\mathsf{st}.nxs$ is applied to a key derivation function KDF_2, in order to produce a new NAXOS string, a KEM key generation randomness r^{KEM}, which is used to produce a new KEM key pair for receiving messages in the next epoch, and a DS key generation r^{DS}, which is used to produce a new DS key pair for sending messages in this epoch. The caller stores the private decapsulation keys and signing keys into the state. Then, the caller remembers the pre-transcript for the ar framework, including the value l, the ciphertext c_1, c_2, and c_3, the sampled encapsulation key ek, and the sampled verification key vk, into the state $\mathsf{st.prtr}$. Afterwards, the caller signs the pre-transcript using the signing key produced two epochs earlier $\mathsf{st.sk}^{\mathsf{st}.t-2}$ for a signature σ^{ar}, followed by appending the signature σ^{ar} into the state pre-transcript $st.\mathsf{prtr}$. Finally, the caller forwards the asymmetric ratchet by applying a KDF_3 to the root key $\mathsf{st}.rk$ and the update $\mathsf{upd}^{\mathsf{ar}}$ for deriving new root key and chain key $\mathsf{st}.ck^{\mathsf{st}.t}$.

Then, the caller executes the so-called *unidirectional rachet* (ur) framework, no matter whether the ar framework is executed in this algorithm invocation or not: First, the caller forwards the unidirectional ratchet chain by applying a KDF_4 to the current chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ for deriving next chain key and a unidirectional ratchet key urk. Next, the caller applies a KDF_5 to the unidirectional ratchet key urk and the update $\mathsf{upd}^{\mathsf{ur}}$ for the message key mk, followed by encrypting the message m by $c' \leftarrow \mathsf{S.Enc}(mk, m)$. Then, the caller sets the pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$ of the ur framework as the epoch $\mathsf{st}.t$, the message index $\mathsf{st}.i$, and the ciphertexts c', c_1, c_2, and c_3. Finally, a signature σ is derived by signing the locally stored ar pre-transcript $\mathsf{st.prtr}$ and the new ur pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$ using the signing key $\mathsf{st.sk}^{\mathsf{st}.t}$.

This algorithm outputs a new state st and a final ciphertext, which is a tuple of the ar pre-transcript $\mathsf{st.prtr}$, the ur pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$, and the signature σ.

$\mathsf{eRcv}(\mathsf{st}, ik, prek, c)$*:* The receiving algorithm inputs the (caller's) state st, the (caller's) private identity key ik and pre-key $prek$, and a ciphertext c, and does the mirror execution of eSend.

First, the caller pares the input ciphertext c into asymmetric ratchet pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$, the unidirectional ratchet pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$, and the signature σ. Next, the caller further pares the pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ into one number l, three ciphertexts c_1, c_2, and c_3, an encapsulation key ek, a verification key vk, and a signature σ^{ar}, and $\mathsf{prtr}^{\mathsf{ur}}$ into an epoch counter t, a message index counter i, and four ciphertexts c', c'_1, c'_2, and c'_3.

If the parsed epoch counter indicates a past epoch, i.e., $t \leq \mathsf{st}.t - 2$, the caller checks whether the maximal transmission length has been set (and not erased) and whether the parsed message index does not exceed the corresponding maximal transmission length. Then, the caller checks whether the parsed epoch counter is valid (by checking whether $\mathsf{st.id} = \mathtt{A}$ or $\mathtt{B}$ if the parsed epoch counter is even or odd) and in a meaningful range (by checking whether $t \leq \mathsf{st}.t + 1$). If any check is wrong, the eRcv aborts and outputs with $m = \perp$. This step in particular prevents the attacker from forging a non-existing message in the previous epochs, as explained in Section C.1.

If the parsed epoch counter t is the next epoch, i.e., $t = \mathsf{st}.t + 1$, the caller executes the asymmetric ratchet framework: The caller first checks whether the signature σ^{ar} is valid under the verification key $\mathsf{st}.vk^{t-2}$ and input $(l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk})$ and aborts if the check fails. Next, the caller invokes $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$, records the transmission length l, and increments the epoch counter. Then, three keys k_1, k_2, and k_3 are respectively decapsulated from c_1, c_2, and c_3 using local keys $\mathsf{st.dk}^{\mathsf{st}.t}$, the private identity key ik, and pre-key $prek$. After that, the caller applies KDF_1 to above keys for update value $\mathsf{upd}^{\mathsf{ar}}$, which then together with the root key $\mathsf{st}.rk$ is applied to KDF_3 for a new root key and chain key $\mathsf{st}.ck^{\mathsf{st}.t}$. Finally, the caller initializes a dictionary $\mathcal{D}^{\mathsf{st}.t}_{urk}$ for storing the unidirectional ratchet keys in this epoch, and the message counter $\mathsf{st}.i$ to 0, and locally stores the encapsulation key for the next epoch and verification key for this epoch.

Then, the caller executes the unidirectional rachet framework, no matter whether the ar framework is executed in this algorithm invocation or not: First, the caller also checks whether the signature σ^{ur} is valid under the verification key $\mathsf{st}.vk^{t}$ and pre-transcripts $(\mathsf{prtr}^{\mathsf{ar}}, \mathsf{prtr}^{\mathsf{ur}})$. Next, three keys k'_1, k'_2, and k'_3 are respectively decapsulated from c'_1, c'_2, and c'_3 using local keys $\mathsf{st.dk}^{\mathsf{st}.t}$, the private identity key ik, and pre-key $prek$. Then, the caller applies KDF_1 to above three keys for the update value $\mathsf{upd}^{\mathsf{ur}}$. After that, the caller continuously forwards the unidirectional ratchet chain, followed by storing the unidirectional ratchet key into the dictionary and incrementing the message index by 1, until the local message index $\mathsf{st}.i$ reaches the parsed message index i. Finally, the caller reads the unidirectional ratchet key urk from the dictionary corresponding to the parsed message index, followed by erasing it from the dictionary, and driving the message key mk by appying KDF_5 to urk and the update $\mathsf{upd}^{\mathsf{ur}}$, and finally decrypts the message m from ciphertext c' using mk.

This algorithm outputs a new state st, the parsed epoch t, the parsed message index i, as well as the decrypted message m.

5.2 Security Conclusion and Concrete Instantiation

We next prove the eSM-*security* of our construction. Our proof is divided into three steps: First, we modularize the eSM-*security* into three simplified security notations: correctness, privacy, and authenticity, which are defined in Supplementary Material C. This step is conceptually similar to the approach of [1], but uses slightly different simplified security notions, because we found a counterexample to their composition result during our investigations. We provide details on the counterexample and our modifications in Supplementary Material C.1.

Next, we reduce the eSM-*security* to the simplified security notions in Theorem 1, the full proof of which is given in Supplementary Material D.1. Finally, we respectively prove the simplified correctness, privacy, and authenticity of our construction in Theorem 2, 3, and 4, the full proof of which are given in Supplementary Material D.2, D.3, and D.4.

Theorem 1. *Let $\mathit{\Pi}$ be an* eSM *scheme that is*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\mathit{\Pi}}^{\mathsf{CORR}})$-CORR *secure,*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\mathit{\Pi}}^{\mathsf{AUTH}})$-AUTH *secure, and*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon_{\mathit{\Pi}}^{\mathsf{PRIV}})$-PRIV *secure*

Then, it is also $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-eSM *secure, where*

$$\epsilon \leq \epsilon_{\mathsf{eSM}}^{\mathsf{CORR}} + q_{\mathsf{ep}}(\epsilon_{\mathsf{eSM}}^{\mathsf{AUTH}} + \epsilon_{\mathsf{eSM}}^{\mathsf{PRIV}})$$

Theorem 2. *Let $\mathit{\Pi}$ denote our* eSM *construction in Section 5.1. If the underlying* KEM*,* DS*, and* SKE *are respectively δ_{KEM}, δ_{DS}, δ_{SKE}-strongly correct*[3] *in time t, then $\mathit{\Pi}$ is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \mathsf{Adv}_{\mathit{\Pi},\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}})$-CORR secure for $\triangle_{\mathsf{eSM}} = 2$, such that*

$$\mathsf{Adv}_{\mathit{\Pi},\triangle_{\mathsf{eSM}}}^{\mathsf{CORR}} \leq (q_{\mathsf{ep}} + q)\delta_{\mathsf{DS}} + 3(q_{\mathsf{ep}} + q)\delta_{\mathsf{KEM}} + q\delta_{\mathsf{SKE}}$$

Theorem 3. *Let $\mathit{\Pi}$ denote our* eSM *construction in Section 5.1. If the underlying* KEM *is $\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$-secure,* SKE *is $\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$-secure,* KDF_1 *is $\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$-secure,* KDF_2 *is $\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$ secure,* KDF_3 *is $\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}$-secure,* KDF_4 *is $\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$-secure,* KDF_5 *is $\epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}}$ and $\epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}$-secure, in time t, then $\mathit{\Pi}$ is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \mathsf{Adv}_{\mathit{\Pi},\triangle_{\mathsf{eSM}}}^{\mathsf{PRIV}})$-PRIV secure for $\triangle_{\mathsf{eSM}} = 2$, such that*

$$\begin{aligned}\mathsf{Adv}_{\mathit{\Pi},\triangle_{\mathsf{eSM}}}^{\mathsf{PRIV}} \leq q\Bigg(& q_{\mathsf{M}}\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + q_{\mathsf{ep}}(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}) + \\ & q_{\mathsf{M}}q_{\mathsf{ep}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}) + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}}\Bigg)\end{aligned}$$

Theorem 4. *Let $\mathit{\Pi}$ denote our* eSM *construction in Section 5.1. If the underlying* DS *is $\epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}$-secure,* KEM *is $\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$-secure,* SKE *is $\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$-secure,* KDF_1 *is $\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$-secure,* KDF_2 *is $\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$ secure,* KDF_3 *is $\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}$-secure,* KDF_4 *is $\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$-secure,* KDF_5 *is $\epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}}$ and $\epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}$-secure, in time t, then $\mathit{\Pi}$ is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \mathsf{Adv}_{\mathit{\Pi},\triangle_{\mathsf{eSM}}}^{\mathsf{AUTH}})$-*AUTH *secure for $\triangle_{\mathsf{eSM}} = 2$, such that*

$$\begin{aligned}\mathsf{Adv}_{\mathit{\Pi},\triangle_{\mathsf{eSM}}}^{\mathsf{AUTH}} \leq & \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}) \\ & + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}} + 1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big)\end{aligned}$$

[3] By strongly correct, we mean that the schemes are conventionally correct for all randomness. See Supplementary Material B for more details.

Remark 1. The strong unforgeability SUF-CMA of the underlying DS is required, since in our model and SM-security model, the attacker can win if the receiver accepts any ciphertext which is not identical to the one produced by the sender. Thus, if the attacker is able to produce a new signature for the original payload, then the attacker still wins. However, we can see that such attack does not enable the attacker to inject any malicious payload and to really interfere the communication channel. We claim that the EUF-CMA is sufficient, if we restrict the attacker to win via $\mathsf{win}^{\mathsf{auth}}$ predicate only when it injects a new message, i.e., the Line 69 in Figure 1 is replaced by $\big(m' \neq \perp$ **and** $(\mathsf{B}, t', i', m') \notin \mathsf{comp}\big)$.

Instantiation: We give the concrete instantiation for both classical and post-quantum setting. The deterministic DS can be instantiated with Ed25519 for classical setting, the formal analysis was given in [12], and the NIST suggested CRYSTALS-Dilithium for the post-quantum security, which is analyzed in [13]. A generic approach to instantiating KEM is to encrypt random string using deterministic OW-CCA or merely OW-CPA secure PKE for strong correctness [14], [15]. The NIST suggested NTRU is also available for IND-CCA security and strong correctness [16]. The deterministic IND-1CCA secure authenticated encryption SKE can be instantiated with the Encrypt-then-MAC construction in [17]. The dual or prg-secure KDF_i for $i \in \{2, ..., 5\}$ can be instantiated with HMAC-SHA256 or HKDF. The 3prf-secure KDF_1 can be instantiated with the nested combination of any dual-secure function F, as explained in Supplementary Material B.4. We suggest to double the security parameter of the symmetric primitives for post-quantum security.

5.3 Comparison to ACD19 and ACD19-PK

Although our eSM construction in Section 5.1 and the ACD19 and ACD19-PK constructions in [1] all satisfy immediate decryption with constant bandwidth consumption, their designs differ in many details.

Comparison between our eSM *construction and* ACD19 : As introduced in Section 2, the ACD19 in [1, Section 5.1] makes use of three underlying modules: CKA, FS-AEAD, and PRF-PRNG. While the CKA employs the asymmetric cryptographic primitives, such as KEM or Diffie-Hellman exchange, the FS-AEAD and PRF-PRNG only employ symmetric cryptographic primitives, such as AEAD, PRF, PRG. In particular, the FS-AEAD deterministically derives the symmetric keys for encrypting messages and decrypting ciphertexts from the state, which is shared by both parties. Besides, they provide several CKA instantiations and all of them samples the asymmetric key pairs only using the ephemeral randomness. Moreover, their construction does not rely on any material outside the session state. Thus, it is easy to see that the leakage of either state will trigger the loss of the privacy and authenticity.

Compared to the ACD19, our eSM construction has the differences mainly from following three aspects: First, the asymmetric primitives are used in every sending or receiving execution. In particular, our construction uses the KEM and DS keys across our asymmetric ratchet (ar) and unidirectional ratchet (ur) frameworks. Although this stops the further modularization of our eSM construction, the deployment of the KEM and DS provides better performance in terms of the strong privacy and strong authenticity, since the leakage of sender's (resp. receiver's) state does not indicates the compromise of the decapsulation key (resp. signing key) and preserves the privacy (resp. authenticity).

Second, our construction employs the NAXOS string, which is used to improve the randomness during the KEM and DS private key generation. Note that the leakage of the ephemeral randomness does not indicate the loss of the privacy of the messages in the current epochs, but might leave potential vulnerability to the future. For instance, if a corrupted party in ACD19 sends a message in a new epoch using fresh randomness, the shared state is *not* healed, if the partner's asymmetric key was generated using predictable randomness. The NAXOS trick in our construction mitigate such attack, since the KEM and DS key pairs are generated using both ephemeral randomness and the *nxs* string in the state.

Finally, our construction makes use of the identity keys and pre-keys, which also provide better performance in terms of the strong privacy, state compromise/failure, and periodic confidentiality recovery. In particular, the corruption of the session state does *not* imply the loss of the privacy, if the sender's randomness is fresh and the receiver's either private identity or pre-key is not leaked.

As an aside, we observe that the CKA instantiation based on LWE (Frodo) does *not* provide correctness: CKA-correctness requires both parties to always output the same key, even if the attacker controls the randomness. Since LWE based Frodo includes an error that needs to be reconciled during the decapsulation, the attacker can always pick bad randomness to prevent the correct reconciliation. Instead, our construction is provably correct in the post-quantum setting, if the underlying KEM satisfies strong correctness, as explained in Section 5.2.

Furthermore, our eSM construction also prevents the forgery attack to which ACD19 is vulnerable, as we show in Supplementary Material C.1.

Comparison between our eSM *construction and* ACD19-PK : The ACD19-PK construction in [1, Section 6.2] is given based on their ACD19 construction. The only difference is that the ACD19-PK additionally employs asymmetric cryptographic primitives PKE and DS. The fresh asymmetric keys for the following epochs are sampled using ephemeral randomness and locally stored right after the execution of CKA. After the execution of FS-AEAD, the sender additionally encrypts the ciphertext output by FS-AEAD using PKE and signs the whole pre-transcript (including the newly generated PKE and DS public keys, and the ciphertext of PKE) using DS. By importing PKE and DS, the strong privacy and strong authenticity are guaranteed.

Although their ACD19-PK construction looks very similar to our eSM construction at the first glance, there do exist many differences. First, our construction employs identity keys and pre-keys outside of the session states as discussed above. This ensures the strong privacy, state compromise/failure, and periodic confidentiality recovery, even when the receiver's session state is compromised. These properties are not satisfied by ACD19-PK.

Second, the ACD19-PK samples KEM and DS for the following stages solely using the ephemeral randomness. As explained above, this does not cause the loss of the privacy in the current epochs but might bring potential vulnerability for the future. Our eSM construction fixed this problem.

Moreover, ACD19-PK samples PKE and DS key pairs only for the future epochs. In particular, when a party starts to send messages at epoch t, its local state includes both the signing key for epoch t and the one for epoch $t+2$. The state exposure means that the attacker can forge messages for both epochs. In other words, the recovery from the session state corruption requires at least four epochs. In our eSM construction, the old signing key will be erased whenever new signing key is generated. This means the healing of our eSM construction still only needs two epochs.

Finally, while ACD19-PK use the "nested encryption" - encrypting the ciphertext of FS-AEAD using PKE, our construction opts to use KEM independent of SKE. This reduces much computational effort and bandwidth, in particular for encrypting large files, which further mitigates the concern in [1, Section 6.1].

Note that the forgery attack mentioned in Supplementary Material C.1 also works against ACD19-PK, but does not work not against our eSM construction.

6 Offline Deniability

As explained in Section 2.2, although the combination of the SPQR and ACD19, ACD19-PK, and our eSM construction achieve strong privacy and authenticity in the post-quantum setting, it is still an open question what flavors of offline deniability can be achieved by the combined protocols in the post-quantum setting. To address this, we first extend the game-based offline deniability for asynchronous DAKE [6] to its combination with an eSM scheme. Then, we prove that the combination of any offline deniable asynchronous DAKE, such as SPQR and our eSM construction in Section 5.1, is offline-deniable in our model. We refer the interested readers to the DAKE scheme and its offline deniability in [6].

Our offline deniability experiment is depicted in Figure 4. The experiment $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,\mathsf{n_P},q_\mathsf{M},\mathsf{n_S}}$ is parameterized by a DAKE scheme Σ, a eSM scheme Π, and the maximal numbers of parties $\mathsf{n_P}$, pre-key per party q_M, and total sessions $\mathsf{n_S}$. For the notational purpose, we use $\overline{ipk}$, $\overline{ik}$, $\overline{prepk}$, and $\overline{prek}$ to denote the public and private keys that are generated by DAKE construction Σ. The keys generated by eSM construction Π are notated without over-line. Due to the page limit, we only explain the difference to the original model in [6, Definition 11], which is highlighted with blue color in Figure 4.

At the beginning of the experiment, the challenger additionally initializes a dictionary $\mathcal{D}_{\mathsf{session}}$, which records the identity of the parties in each session, and a session counter n with 0. Next, the challenger respectively generates the public-private identity key pairs and pre-key pairs by running $\Pi.\mathsf{IdKGen}$ and $\Pi.\mathsf{PreKGen}$. All of these eSM key pairs are added into a list $\mathcal{L}_{\mathsf{all}}$, which records all public and private key pairs in this experiment. For each party u, the challenger also sets up a list $\mathcal{L}_u^{prek}$ that records all private pre-key of u. Then, the attacker is given the list $\mathcal{L}_{\mathsf{all}}$ and the access to two oracles and wins if it can guess the challenge bit correctly.

$\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$: This oracle inputs a sender identity sid, a receiver identity rid, a accuser identity aid, a defendant identity did, and a pre-key index ind. This oracle first checks whether the sender identity and the receiver identity are distinct and whether either the sender is the accuser and the receiver is the defendant or another way around. Next, increments the session counter n by 1 and sets the set of the sender identity sid and the receiver identity rid to $\mathcal{D}_{\mathsf{session}}[i]$. Then, it simulates the honest AKE execution if the challenge bit is 0 or the accuser is the sender. Otherwise, it runs the fake algorithm $\Sigma.\mathsf{Fake}$. In both cases, a key K and a transcript T are derived. In the end, if the challenge bit is 0, then the oracle honestly runs $\Pi.\mathsf{eInit\text{-}A}(K)$ and $\Pi.\mathsf{eInit\text{-}B}(K)$ on the shared key K to produce the state $\mathsf{st}^n_{\mathsf{sid}}$ and $\mathsf{st}^n_{\mathsf{rid}}$. Otherwise, the oracle runs a function $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}^{prek}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ to produce a fake state $\mathsf{st}^n_{\mathsf{Fake}}$. The transcript T is returned.

$\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$: This oracle inputs a sender identity sid, a receiver identity rid, a session index sessID, a pre-key index ind, and a message m. This oracle first checks whether the session between sid and rid has been established by requiring $\mathcal{D}_{\mathsf{session}}[i] = \{\mathsf{sid}, \mathsf{rid}\}$. Next, if the challenge bit is 0, this oracle simulates the honest transmission of message m and outputs the transmitted ciphertext c. Otherwise, this oracle produces a ciphertext c by running a function $\mathsf{Fake}^{\mathsf{eSend}}_{\Pi}$ on the fake state $\mathsf{st}^i_{\mathsf{Fake}}$, the receiver's public identity key ipk_{rid}, pre-key $prepk^{\mathsf{ind}}_{\mathsf{rid}}$, the message m, and sender identity sid, the receiver identity rid, and a pre-key index ind, followed by returning the produced ciphertext c.

We stress that our offline deniability model is a significant extension to the one for AKE in [6, Definition 11]. While the offline deniability in [6, Definition 11] prevents a judge from distinguishing the real key establishment transcript from a frame-up by the accuser without the participation of the innocent defendant, our model captures that the real transcript of the combination of a DAKE scheme and an eSM scheme is indistinguishable from the frame-up by the accuser, without the participation of the innocent defendant, no matter the accuser is the sender (aka. session initiator) or the receiver (aka. session responder) in the AKE.

Definition 5. *We say the composition of a* DAKE *scheme* Σ *and an* eSM *scheme* Π *is* $(t, \epsilon, \mathsf{n_P}, q_{\mathsf{M}}, \mathsf{n_S})$*-deniable, if there exist two functions* $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}$ *and* $\mathsf{Fake}^{\mathsf{eSend}}_{\Pi}$ *such that the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}^{\mathsf{deni}}_{\Sigma,\Pi} := |\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,\mathsf{n_P},q_{\mathsf{M}},\mathsf{n_S}}(\mathcal{A}) - \frac{1}{2}| \leq \epsilon$$

where $\mathsf{n_P}$*,* q_{M}*, and* $\mathsf{n_S}$ *respectively denote the maximal number of parties, of pre-key per party, and the total session in the* $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,\mathsf{n_P},q_{\mathsf{M}},\mathsf{n_S}}$ *in Figure 4.*

Theorem 5. *Let* Σ *denote a* DAKE *scheme and* Π *denote our* eSM *construction in Section 5.1. If* Σ *is* (t, ϵ, q)*-deniable in terms of the [6, Definition 11], then the composition of* Σ *and* Π *is* $(t, \epsilon, \mathsf{n_P}, q_{\mathsf{M}}, q)$*-deniable for all* $\mathsf{n_P}$*,* q_{M}*.*

We give the proof in Section Supplementary Material D.5.

```
$\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,\mathsf{n_P},q_\mathsf{M},\mathsf{n_S}}(\mathcal{A})$:
$\mathcal{D}_{\mathsf{session}}[\cdot] \leftarrow \perp,\ n \leftarrow 0$
$\mathcal{L}_{\mathsf{all}}, \mathcal{L}_{\mathsf{all}}^{\overline{ipk}}, \mathcal{L}_{\mathsf{all}}^{\overline{prepk}} \leftarrow \emptyset$
for $u \in [\mathsf{n_P}]$
    $\mathcal{L}_u^{\overline{prek}} \leftarrow \emptyset$
    $\mathcal{L}_u^{prek} \leftarrow \emptyset$
    $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$
    $(ipk_u, ik_u) \xleftarrow{\$} \Pi.\mathsf{IdKGen}()$
    $\mathcal{L}_{\mathsf{all}}^{\overline{ipk}} \xleftarrow{+} \{\overline{ipk}_u\}$
    $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{ipk}_u, \overline{ik}_u)$
    $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (ipk_u, ik_u)$
    for $\mathsf{ind} \in [q_\mathsf{M}]$
        $(\overline{prepk}_u^{\mathsf{ind}}, \overline{prek}_u^{\mathsf{ind}}) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$
        $(prepk_u^{\mathsf{ind}}, prek_u^{\mathsf{ind}}) \xleftarrow{\$} \Pi.\mathsf{PreKGen}()$
        $\mathcal{L}_u^{\overline{prek}} \xleftarrow{+} \overline{prek}_u^{\mathsf{ind}},\ \mathcal{L}_{\mathsf{all}}^{\overline{prepk}} \xleftarrow{+} \overline{prepk}_u^{\mathsf{ind}}$
        $\mathcal{L}_u^{prek} \xleftarrow{+} prek_u^{\mathsf{ind}}$
        $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{prepk}_u, \overline{prek}_u)$
        $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (prepk_u, prek_u)$
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathcal{L}_{\mathsf{all}})$
return $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$:
req $\{\mathsf{aid}, \mathsf{did}\} = \{\mathsf{sid}, \mathsf{rid}\}$ and $\mathsf{sid} \neq \mathsf{rid}$
$n{+}{+},\ \mathcal{D}_{\mathsf{session}}[n] \leftarrow \{\mathsf{sid}, \mathsf{rid}\}$
if $\mathsf{b} = 0$ or $\mathsf{aid} = \mathsf{sid}$
    $\pi_{\mathsf{rid}}.role \leftarrow \mathtt{resp},\ \pi_{\mathsf{rid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $\pi_{\mathsf{sid}}.role \leftarrow \mathtt{init},\ \pi_{\mathsf{sid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $(\pi'_{\mathsf{rid}}, m) \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{rid}}, \mathcal{L}_{\mathsf{rid}}^{\overline{prek}}, \mathcal{L}_{\mathsf{all}}^{\overline{ipk}}, \mathcal{L}_{\mathsf{all}}^{\overline{prepk}}, \pi_{\mathsf{rid}}, (\mathsf{create}, \mathsf{ind}))$
    $(\pi'_{\mathsf{sid}}, m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{sid}}, \mathcal{L}_{\mathsf{sid}}^{\overline{prek}}, \mathcal{L}_{\mathsf{all}}^{\overline{ipk}}, \mathcal{L}_{\mathsf{all}}^{\overline{prepk}}, \pi_{\mathsf{sid}}, m)$
    $(K, T) \xleftarrow{\$} (\pi'_{\mathsf{sid}}.K, (m, m'))$
else
    $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_{\mathsf{sid}}, \overline{ik}_{\mathsf{rid}}, \mathcal{L}_{\mathsf{rid}}^{\overline{prek}}, \mathsf{ind})$
if $\mathsf{b} = 0$
    $\mathsf{st}_{\mathsf{sid}}^n \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}B}(K),\ \mathsf{st}_{\mathsf{rid}}^n \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}A}(K)$
else
    $\mathsf{st}_{\mathsf{Fake}}^n \xleftarrow{\$} \mathsf{Fake}_{\Pi}^{\mathsf{eInit}}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}_{\mathsf{aid}}^{prek}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$
return $T$

$\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$:
req $\mathcal{D}_{\mathsf{session}}[i] = \{\mathsf{sid}, \mathsf{rid}\}$
if $\mathsf{b} = 0$
    $(\mathsf{st}_{\mathsf{sid}}^i, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}_{\mathsf{sid}}^i, ipk_{\mathsf{rid}}, prepk_{\mathsf{rid}}^{\mathsf{ind}}, m)$
    $(\mathsf{st}_{\mathsf{rid}}^i, \_, \_, \_) \leftarrow \Pi.\mathsf{eRcv}(\mathsf{st}_{\mathsf{rid}}^i, ik_{\mathsf{rid}}, prek_{\mathsf{rid}}^{\mathsf{ind}}, c)$
else
    $(\mathsf{st}_{\mathsf{Fake}}^i, c) \xleftarrow{\$} \mathsf{Fake}_{\Pi}^{\mathsf{eSend}}(\mathsf{st}_{\mathsf{Fake}}^i, ipk_{\mathsf{rid}}, prepk_{\mathsf{rid}}^{\mathsf{ind}}, m, \mathsf{sid}, \mathsf{rid}, \mathsf{ind})$
return $c$
```

Fig. 4: The offline deniability experiment for an attacker $\mathcal{A}$ against the combination of a DAKE scheme Σ and an eSM scheme Π. We highlight the difference to the offline deniability experiment for AKE in [6, Definition 11] with blue color.

References

[1] J. Alwen, S. Coretti, and Y. Dodis, “The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol,” in *Advances in Cryptology – EUROCRYPT 2019*, Springer, 2019.

[2] J. Jaeger and I. Stepanovs, “Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging,” in *Advances in Cryptology – CRYPTO 2018*, Springer, 2018.

[3] B. Poettering and P. Rösler, “Towards Bidirectional Ratcheted Key Exchange,” in *Advances in Cryptology – CRYPTO 2018*, Springer, 2018.

[4] D. Jost, U. Maurer, and M. Mularczyk, “Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging,” in *Advances in Cryptology – EUROCRYPT 2019*, Springer, 2019.

[5] F. B. Durak and S. Vaudenay, “Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity,” in *Advances in Information and Computer Security*, Springer, 2019.

[6] J. Brendel, R. Fiedler, F. Günther, C. Janson, and D. Stebila, *Post-quantum asynchronous deniable key exchange and the signal handshake*, Cryptology ePrint Archive, Report 2021/769, `https://ia.cr/2021/769`, 2021.

[7] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila, “A Formal Security Analysis of the Signal Messaging Protocol,” in *EuroS&P*, IEEE, 2017.

[8] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, “A Formal Security Analysis of the Signal Messaging Protocol,” *Journal of Cryptology*, vol. 33, no. 4, 2020.

[9] N. Vatandas, R. Gennaro, B. Ithurburn, and H. Krawczyk, “On the Cryptographic Deniability of the Signal Protocol,” in *Applied Cryptography and Network Security*, Springer, 2020.

[10] M. Di Raimondo, R. Gennaro, and H. Krawczyk, “Deniable Authentication and Key Exchange,” in *ACM CCS*, ACM, 2006.

[11] K. Hashimoto, S. Katsumata, K. Kwiatkowski, and T. Prest, “An Efficient and Generic Construction for Signal’s Handshake (X3DH): Post-Quantum, State Leakage Secure, and Deniable,” in *Public-Key Cryptography - PKC 2021*, Springer, 2021.

[12] J. Brendel, C. Cremers, D. Jackson, and M. Zhao, “The provable security of ed25519: Theory and practice,” in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021.

[13] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, *CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation (Version 3.1)*, https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[14] M. Bellare, M. Fischlin, A. O’Neill, and T. Ristenpart, “Deterministic encryption: Definitional equivalences and constructions without random oracles,” in *Annual International Cryptology Conference*, Springer, 2008.

[15] D. J. Bernstein and E. Persichetti, *Towards KEM unification*, Cryptology ePrint Archive, Paper 2018/526, https://eprint.iacr.org/2018/526.

[16] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, and Z. Zhang, “Algorithm specifications and supporting documentation,” *Brown University and Onboard security company, Wilmington USA*, 2019.

[17] M. Bellare and C. Namprempre, “Authenticated encryption: Relations among notions and analysis of the generic composition paradigm,” in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2000.

Supplementary Material

A Review on "optimal" and "sub-optimal" secure protocols

The "optimal" protocols by Jäger and Stepanovs [2] and by Pöttering Rösler [3] and the "sub-optimal" protocol by Durak and Vaudenay [5] all are post-quantum compatible. Technically, they follow different ratcheting frameworks:

1. **"optimal" Jäger-Stepanovs protocol [2]**: In the Jäger-Stepanovs protocol, all cryptographic building blocks except the hash functions, such as PKE and DS, are asymmetric and updatable. When Alice continuously sends messages to Bob, the next encryption key is deterministically derived from an encryption key included in the last reply from Bob and all past transcript since the last reply from Bob. On the one hand, this protocol enjoys high security guarantee against impersonation due to the asymmetric state. On the other hand, this protocol has no message-loss resilience, namely, if one message from Alice to Bob is lost, then Bob cannot decrypt subsequent messages anymore. In particular, no instantiation with constant bandwidth in the post-quantum setting is available.
2. **"optimal" Pöttering-Rösler protocol [3]**: In the Pöttering-Rösler protocol, both asymmetric and symmetric primitives, including updatable KEM, DS, MAC are employed. When Alice sends messages to Bob, she first runs the encapsulations upon the one or more KEM public keys depending on her behavior. If Alice is sending a reply, then she needs to run the encapsulation upon all accumulated KEM public keys that are generated and signed by Bob. Otherwise, she only needs one KEM public key that was generated by herself when sending the previous message. After that, Alice derives the symmetric key for message encryption from the symmetric state and the encapsulated keys. This protocol enjoys *state healing* when continuously sending messages. Any unpredictable randomness at some point can heal Alice's state from corruption when she continuously sends messages. However, this protocol has no message-loss resilience: If one message is lost in the transmission, the both parties' symmetric states that are used for key update mismatch. This means, all subsequent messages cannot be correctly recovered by the recipient.
3. **"sub-optimal" Durak-Vaudenay protocol [5]**: In contrast to the above two "optimal" approaches, the Durak-Vaudenay protocol does not employ any key updatable components and has a substantially better time complexity. When Alice sends messages to Bob, she samples several fragments of a symmetric key and encrypts them using signcryption with the accumulated sender keys, where the sender keys are generated either by herself or by Bob depending on whether Alice is continuously sending messages or sending a reply. The Durak-Vaudenay protocol is similar to Pöttering-Rösler but is less reliant on the state. Any randomness leakage corrupts the next message. Moreover, both the message and the receiver key that is used for receiving or sending next message, are encrypted under the symmetric key. This implies that the protocol does not have message-loss resilience: If one message is lost

in the transmission (from either Alice or Bob), the communication session is aborted.

Thus, none of these three protocols provide message-loss resilience and all of them require linearly growing memory and dynamic bandwidth.

B Preliminaries

B.1 Key Encapsulation Mechanisms

Definition 6. *A key encapsulation mechanism (*KEM*) scheme over randomness space* $\mathcal{R}$ *and symmetric key space* $\mathcal{K}$ *is a tuple of algorithms* $\mathsf{KEM} = (\mathsf{K.KG}, \mathsf{K.Enc}, \mathsf{K.Dec})$ *as defined below.*

- ***Key Generation*** $(\mathsf{ek}, \mathsf{dk}) \overset{\$}{\leftarrow} \mathsf{K.KG}(\mathsf{pp})$*: takes as input the public parameter* pp *and outputs a public encapsulation and private decapsulation key pair* $(\mathsf{ek}, \mathsf{dk})$ *.*
- ***Encapsulation*** $(c, k) \overset{\$}{\leftarrow} \mathsf{K.Enc}(\mathsf{ek})$*: takes as input a public key* pk *and outputs a ciphertext* c *and a symmetric key* k*. We write* $(c, k) \overset{\$}{\leftarrow} \mathsf{K.Enc}(\mathsf{ek}; r^{\mathsf{Encaps}})$ *if the random coins* $r^{\mathsf{Encaps}} \in \mathcal{R}$ *is specified.*
- ***Decapsulation*** $k \leftarrow \mathsf{K.Dec}(\mathsf{dk}, c)$*: takes as input a secret key* dk *and a ciphertext* c *and outputs either a symmetric key* k *or an error symbol* $\perp$*.*

We say a KEM is δ-correct if for every $(\mathsf{ek}, \mathsf{dk}) \overset{\$}{\leftarrow} \mathsf{K.KG}()$, we have

$$\Pr[k \neq \mathsf{K.Dec}(\mathsf{dk}, c) : (c, k) \overset{\$}{\leftarrow} \mathsf{K.Enc}(\mathsf{ek})] \leq \delta$$

In particular, we call a KEM *(perfectly) correct* if $\delta = 0$.

We say a KEM is δ-strongly correct if for every $(\mathsf{ek}, \mathsf{dk}) \overset{\$}{\leftarrow} \mathsf{K.KG}()$ and every $r^{\mathsf{Encaps}} \in \mathcal{R}$, we have

$$\Pr[k \neq \mathsf{K.Dec}(\mathsf{dk}, c) : (c, k) \overset{\$}{\leftarrow} \mathsf{K.Enc}(\mathsf{ek}; r^{\mathsf{Encaps}})] \leq \delta$$

Compared to the conventional correctness, the strong correctness requires that the encapsulate keys can be correctly recovered for every randomness coins involved during the encapsulation. In particular, we call a KEM *(perfectly) strongly correct* if $\delta = 0$.

In terms of the security notions, we recall the standard *indistinguishability under chosen plaintext/ciphertext attacks* ($\mathsf{IND\text{-}CPA}$/$\mathsf{IND\text{-}CCA}$). The $\mathsf{IND\text{-}CPA}$ security prevents an attacker from distinguishing the encapsulated symmetric key of a challenge ciphertext from a random one. The $\mathsf{IND\text{-}CCA}$ security additionally allows the attacker to access a decapsulation oracle.

Definition 7. *Let* $\mathsf{KEM} = (\mathsf{K.KG}, \mathsf{K.Enc}, \mathsf{K.Dec})$ *be a key encapsulation mechanism scheme with symmetric space* $\mathcal{K}$*. We say* KEM *is* ϵ-$\mathsf{IND\text{-}XXX}$ *secure for* $\mathsf{XXX} \in \{\mathsf{CPA}, \mathsf{CCA}\}$*, if for every (potential quantum) adversary* $\mathcal{A}$ *, we have*

$$\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) := \left| \Pr[\mathrm{Expt}_{\mathsf{KEM}}^{\mathsf{IND\text{-}XXX}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

where the $\mathrm{Expt}_{\mathsf{KEM}}^{\mathsf{IND\text{-}XXX}}(\mathcal{A})$ *experiment is defined in Figure 5.*

```
$\mathrm{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM}}(\mathcal{A})$:
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{K.KG}()$
    $(c^\star, k_0^\star) \xleftarrow{\$} \mathsf{K.Enc}(\mathsf{ek})$
    $k_1^\star \xleftarrow{\$} \mathcal{K}$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}(\mathsf{ek}, c^\star, k_\mathsf{b}^\star)$
    return $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathrm{Expt}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}(\mathcal{A})$:
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{K.KG}()$
    $(c^\star, k_0^\star) \xleftarrow{\$} \mathsf{K.Enc}(\mathsf{ek})$
    $k_1^\star \xleftarrow{\$} \mathcal{K}$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Decaps}}}(\mathsf{ek}, c^\star, k_\mathsf{b}^\star)$
    return $[\![\mathsf{b} = \mathsf{b}']\!]$

$\mathcal{O}_{\mathsf{Decaps}}(c)$:
    if $c = c^\star$
        return $\perp$
    $k' \leftarrow \mathsf{K.Dec}(\mathsf{dk}, c)$
    return $k'$
```

Fig. 5: IND-CPA and IND-CCA experiments for $\mathsf{KEM} = (\mathsf{K.KG}, \mathsf{K.Enc}, \mathsf{K.Dec})$ with symmetric key space $\mathcal{K}$.

B.2 Digital Signature

Definition 8. *A digital signature scheme over message space $\mathcal{M}$ and randomness space $\mathcal{R}$ is a tuple of algorithms $\mathsf{DS} = (\mathsf{D.KG}, \mathsf{D.Sign}, \mathsf{D.Vrfy})$ as defined below.*

- ***Key Generation*** $(\mathsf{vk}, sk) \xleftarrow{\$} \mathsf{D.KG}(\mathsf{pp})$*: inputs the public parameter* pp *and outputs a public verification and private signing key pair* (vk, sk)*.*
- ***Signing*** $\sigma \xleftarrow{\$} \mathsf{D.Sign}(sk, m; r^{\mathsf{Sign}})$*: inputs a signing key sk and a message $m \in \mathcal{M}$ and outputs a signature σ; if the random coins $r^{\mathsf{Sign}} \in \mathcal{R}$ is specified.*
- ***Verification*** true/false $\leftarrow \mathsf{D.Vrfy}(\mathsf{vk}, m, \sigma)$*: inputs a verification key* vk*, a message m, and a signature σ and outputs a boolean value either true* true *or* false*.*

We say a DS is δ-correct if for every $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{D.KG}()$ and every message $m \in \mathcal{M}$, we have

$$\Pr[\text{false} \leftarrow \mathsf{D.Vrfy}(\mathsf{vk}, m, \mathsf{D.Sign}(\mathsf{sk}, m))] \leq \delta$$

In particular, we call a DS *(perfectly) correct* if $\delta = 0$.

We say a DS is δ-strongly correct if for every $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{D.KG}()$, every message $m \in \mathcal{M}$, and every $r^{\mathsf{Sign}} \in \mathcal{R}$ we have

$$\Pr[\text{false} \leftarrow \mathsf{D.Vrfy}(\mathsf{vk}, m, \mathsf{D.Sign}(\mathsf{sk}, m; r^{\mathsf{Sign}}))] \leq \delta$$

Compared to the conventional correctness, the strong correctness requires that the signed message-signature pair be correctly verified for every randomness coins involved during the signing. In particular, we call a DS *(perfectly) strongly correct* if $\delta = 0$.

In terms of the security notations, we recall the standard *(strongly) existential unforgeability against chosen message attack* $\mathsf{EUF\text{-}CMA}$ and $\mathsf{SUF\text{-}CMA}$.

Definition 9. *Let $\mathsf{DS} = (\mathsf{D.KG}, \mathsf{D.Sign}, \mathsf{K.Dec})$ be a digital signature scheme with message space $\mathcal{M}$. We say DS is ϵ-$\mathsf{EUF\text{-}CMA}$ secure (resp. ϵ-$\mathsf{SUF\text{-}CMA}$ secure), if for every (potential quantum) adversary $\mathcal{A}$, we have*

$$\epsilon^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) := \Pr[\mathrm{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) = 1] \leq \epsilon$$

$$\epsilon^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) := \Pr[\mathrm{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A}) = 1] \leq \epsilon$$

where the experiment $\mathrm{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$ *and* $\mathrm{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$ *are defined in Figure 6.*

```
$\mathrm{Expt}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$:
1  $\mathcal{L} \leftarrow \emptyset$
2  $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{D.KG}()$
3  $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{vk})$
4  if $m^\star \in \mathcal{L}$
5      return $0$
6  return $[\![\mathsf{D.Vrfy}(\mathsf{vk}, m^\star, \sigma^\star)]\!]$
$\mathcal{O}_{\mathsf{Sign}}(m)$:
7  $\sigma \xleftarrow{\$} \mathsf{D.Sign}(\mathsf{sk}, m)$
8  $\mathcal{L} \xleftarrow{+} m$
9  return $\sigma$
```

```
$\mathrm{Expt}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}(\mathcal{A})$:
1  $\mathcal{L} \leftarrow \emptyset$
2  $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{D.KG}()$
3  $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{vk})$
4  if $(m^\star, \sigma^\star) \in \mathcal{L}$
5      return $0$
6  return $[\![\mathsf{D.Vrfy}(\mathsf{vk}, m^\star, \sigma^\star)]\!]$
$\mathcal{O}_{\mathsf{Sign}}(m)$:
7  $\sigma \xleftarrow{\$} \mathsf{D.Sign}(\mathsf{sk}, m)$
8  $\mathcal{L} \xleftarrow{+} (m, \sigma)$
9  return $\sigma$
```

Fig. 6: $\mathsf{EUF\text{-}CMA}$ and $\mathsf{SUF\text{-}CMA}$ experiments for $\mathsf{DS} = (\mathsf{D.KG}, \mathsf{D.Sign}, \mathsf{D.Vrfy})$.

B.3 Authenticated Encryption

Definition 10. *An authenticated encryption (*SKE*) scheme over message space* $\mathcal{M}$*, randomness space* $\mathcal{R}$*, symmetric key space* $\mathcal{K}$*, and ciphertext space* $\mathcal{C}$ *is a tuple of algorithms* $\mathsf{SKE} = (\mathsf{S.KG}, \mathsf{S.Enc}, \mathsf{S.Dec})$ *as defined below.*

- ***Key Generation*** $k \xleftarrow{\$} \mathsf{S.KG}(\mathsf{pp})$*: takes as input the public parameter* pp *and outputs a symmetric key* $k \in \mathcal{K}$*.*
- ***Encryption*** $c \xleftarrow{\$} \mathsf{S.Enc}(k, m; r^{\mathsf{Enc}})$*: takes as input a symmetric key* k *and a message* m *and outputs a ciphertext* c*. We write* $c \xleftarrow{\$} \mathsf{S.Enc}(k; r^{\mathsf{Enc}})$ *if the random coins* $r^{\mathsf{Enc}} \in \mathcal{R}$ *is specified.*
- ***Decryption*** $m \leftarrow \mathsf{S.Dec}(k, c)$*: takes as input a symmetric key* k *and a ciphertext* c *and outputs either a symmetric key* k *or an error symbol* $\perp$*.*

We say a SKE is δ-correct if for every $k \xleftarrow{\$} \mathsf{S.KG}()$ and every message $m \in \mathcal{M}$, we have

$$\Pr[m \neq \mathsf{S.Dec}(k, \mathsf{S.Enc}(k, m))] \leq \delta$$

In particular, we call a SKE *(perfectly) correct* if $\delta = 0$.

We say a SKE is δ-correct if for every $k \xleftarrow{\$} \mathsf{S.KG}()$, every message $m \in \mathcal{M}$,and every $r^{\mathsf{Enc}} \in \mathcal{R}$, we have

$$\Pr[m \neq \mathsf{S.Dec}(k, \mathsf{S.Enc}(k, m; r^{\mathsf{Enc}}))] \leq \delta$$

Compared to the conventional correctness, the strong correctness requires that the encrypted message can be correctly recovered for every randomness coins involved during the encryption. In particular, we call a SKE *(perfectly) strongly correct* if $\delta = 0$.

In terms of the security notions, we recall the *indistinguishability under one-time chosen ciphertext attacks* (IND-1CCA). In this security notion, the attacker is allowed to query the encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ at most once. However, the attacker can have access to the decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ with arbitrary times.

In particular, this security notion is achievable even for deterministic SKE.

Definition 11. *Let* $\mathsf{SKE} = (\mathsf{S.KG}, \mathsf{S.Enc}, \mathsf{S.Dec})$ *be an authenticated encryption scheme with ciphertext space* $\mathcal{C}$. *We say* SKE *is* ϵ-IND-1CCA *secure, if for every (potential quantum) adversary* $\mathcal{A}$ *, we have*

$$\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{A}) := \left| \Pr[\mathrm{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{A}) = 1] - \frac{1}{2} \right| \leq \epsilon$$

where the $\mathrm{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{A})$ *experiment is defined in Figure 7.*

```
$\mathrm{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}(\mathcal{A})$:
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$k \xleftarrow{\$} \mathsf{S.KG}()$
$c^\star \leftarrow \perp$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}}()$
return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\mathcal{O}_{\mathsf{Enc}}(m)$:
req $c^\star = \perp$
if $\mathsf{b} = 0$
    $c^\star \xleftarrow{\$} \mathsf{S.Enc}(k, m)$
else
    $c^\star \xleftarrow{\$} \mathcal{C}$
return $c$
```

```
$\mathcal{O}_{\mathsf{Dec}}(c)$:
if $c = c^\star$ or $\mathsf{b} = 1$
    return $\perp$
return $\mathsf{S.Dec}(k, c)$
```

Fig. 7: IND-1CCA experiment for $\mathsf{SKE} = (\mathsf{S.KG}, \mathsf{S.Enc}, \mathsf{S.Dec})$ with ciphertext space $\mathcal{C}$.

B.4 Pseudorandom Generators and Pseudorandom Functions

Definition 12. *Let* $\mathsf{F} : \mathcal{R} \rightarrow \mathcal{O}$ *denote a function that maps a random string* $r \in \mathcal{R}$ *to an output* $y \in \mathcal{O}$. *We say* F *is* ϵ-prg *secure if for any variable* X *that follows uniform distribution over* $\mathcal{R}$ *and any variable* Y *that follows uniform distribution over* $\mathcal{O}$*, we have*

$$\mathsf{Adv}_{\mathsf{F}}^{\mathsf{prg}}(\mathcal{D}) := \left| \Pr[\mathcal{D}(\mathsf{F}(X)) = 1] - \Pr[\mathcal{D}(Y) = 1] \right| \leq \epsilon$$

Definition 13. *Let* $\mathsf{F} : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{O}$ *be a function that maps a key* $k \in \mathcal{K}$ *and a string* $m \in \mathcal{M}$ *to an output* $y \in \mathcal{O}$. *We say* F *is* ϵ-prf*-secure if for any* $k \xleftarrow{\$} \mathcal{K}$ *and any truly random function* $\mathbf{R} : \mathcal{M} \rightarrow \mathcal{O}$*, we have*

$$\mathsf{Adv}_{\mathsf{F}}^{\mathsf{prf}}(\mathcal{D}) := \left| \Pr[\mathcal{D}^{\mathsf{F}(k,\cdot)} = 1] - \Pr[\mathcal{D}^{\mathbf{R}(\cdot)} = 1] \right| \leq \epsilon$$

We say PRF *is* swap*-secure if the argument-swapped function* $\bar{\mathsf{PRF}}(m,k) := \mathsf{PRF}(k,m)$ *is* prf*-secure. We say* PRF *is a* dual*-PRF when it is both* prf*-secure and* swap*-secure.*

Definition 14. *Let* $m \geq 2$*. Let* $\mathsf{F} : \mathcal{K}_1 \times ... \times \mathcal{K}_m \to \mathcal{O}$ *be a function that maps* m *keys* $k_i \in \mathcal{K}_i$ *for* $1 \leq i \leq m$ *to an output* $y \in \mathcal{O}$*. We say* F *is* ϵ*-*mprf*-secure if all of the functions* $\bar{\mathsf{F}}_i(k_i,(k_1,...,k_{i-1},k_{i+1},...,k_m)) := \mathsf{F}(k_1,...,k_m)$ *is* prf*-secure.*

The mprf secure function can be easily construction from dual-secure functions. In this paper, we makes use of a mprf-secure KDF for $m = 3$. Below, we present the instantiation and prove the security.

Theorem 6. *Let* $\mathsf{F}_1 : \mathcal{K}_1 \times \mathcal{K}_2 \to \mathcal{O}_1$ *and* $\mathsf{F}_2 : \mathcal{O}_1 \times \mathcal{K}_3 \to \mathcal{O}_2$ *be two functions. If* F_1 *and* F_2 *both are* ϵ*-*dual*-secure, then the function* $\mathsf{F}'(k_1,k_2,k_3) := \mathsf{F}_2(\mathsf{F}_1(k_1,k_2),k_3)$ *is* ϵ'*-*$\mathsf{3prf}$*-secure such that* $\epsilon' \leq q\epsilon$*, where* q *denotes the number of queries by any attacker against* $\mathsf{3prf}$*-security of* F'*.*

Proof. We first show that $\bar{\mathsf{F}}_1(k_1,(k_2,k_3)) := \mathsf{F}'(k_1,k_2,k_3) = \mathsf{F}_2(\mathsf{F}_1(k_1,k_2),k_3)$ is prf-secure. We prove this by game hopping. Let q denote the number of queries that an attacker $\mathcal{A}$ makes. Let Adv_i denote the advantage of $\mathcal{A}$ in winning game i.
Game 0. This game is identical to the experiment. And we have that $\mathsf{Adv}_0 := \epsilon'$
Game 1. In this game, whenever $\mathcal{A}$ queries (k_2,k_3), the challenger samples a random y_1 and replaces $\bar{\mathsf{F}}_1(k_1,(k_2,k_3)) = \mathsf{F}_2(\mathsf{F}_1(k_1,k_2),k_3)$ by $\bar{\mathsf{F}}_1(k_1,(k_2,k_3)) = \mathsf{F}_2(y_1,k_3)$. If the attacker $\mathcal{A}$ can distinguish **Game 0** and **Game 1**, then we can easily construct an attacker that breaks the prf security of F_1. Thus, $\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq \epsilon$.
Game 2. In this game, whenever $\mathcal{A}$ queries (k_2,k_3), the challenger samples a random y_1 and replaces $\bar{\mathsf{F}}_1(k_1,(k_2,k_3)) = \mathsf{F}_2(y_1,k_3)$ by $\bar{\mathsf{F}}_1(k_1,(k_2,k_3)) = y_2$.

If the attacker $\mathcal{A}$ can distinguish **Game 0** and **Game 1**, then we can easily construct an attacker that breaks the prf security of at least one of q F_2. Thus, $\mathsf{Adv}_0 - \mathsf{Adv}_1 \leq q\epsilon$.

Now, in **Game 2** the challenger always simulates the random function. Thus, $\mathcal{A}$ cannot distinguish it, and we have that $\epsilon \leq (q+1)\epsilon$.

The analysis for the prf-security of $\bar{\mathsf{F}}_2(k_2,(k_1,k_3))$:=$\mathsf{F}'(k_1,k_2,k_3) = \mathsf{F}_2(\mathsf{F}_1(k_1, k_2),k_3)$ and $\bar{\mathsf{F}}_3(k_3,(k_1,k_2))$:=$\mathsf{F}'(k_1,k_2,k_3)$= $\mathsf{F}_2(\mathsf{F}_1(k_1,k_2),k_3)$ is similar.

C Security Modularization

The analysis for the security of messaging protocols are often very tedious, since both the security model and the protocols are usually highly complex. Alwen et al. [1] opt to first reduce the SM-security into several simplified security notions: correctness, privacy, and authenticity. Then, they respectively prove the individual simplified security of their proposal $\mathsf{ACD19}$. However, we find out a technical flaw in their reduction, which indicates that their proposal satisfies each of their simplified security notions but *not* the original one.

We adopt the same strategy. In Section C.1, we first explain the technical flaw in [1], which is entailed by their inappropriate definition of the simplified authenticity experiment. In Section C.2, we split the eSM-security into several new simplified security notions and prove the reduction between eSM and the new simplified security notions.

C.1 Flaws in [1]

In order to present the flaws in [1], we first briefly recall the related oracles in the full SM-security model against a secure messaging scheme $\mathsf{SM} = (\mathsf{Init\text{-}A}, \mathsf{Init\text{-}B}, \mathsf{Send}, \mathsf{Rcv})$ and the modification in the simplified authenticity game.

Background: In the SM-security game, the attacker is allowed to query injection oracles INJECT-A to deliver a forged ciphertext c to party $\mathtt{A}$. First, the INJECT-A oracle aborts if $(\mathtt{B}, c)$ is included in the transcript set or the safe injection predicate safe-inj is false. In SM-security game, the safe-inj predicate is true if and only if the state of both parties $\mathtt{A}$ and $\mathtt{B}$ are not leaked within $\triangle_{\mathsf{SM}}$ epochs since the last state corruption of either party. Next, Rcv algorithm inputs party $\mathtt{A}$'s state $\mathsf{st}_{\mathtt{A}}$ and the ciphertext c and outputs a new state $\mathsf{st}_{\mathtt{A}}$, an epoch t', a message index i', and a message m'. If $m' \neq \perp$ and $(\mathtt{B}, t', i')$ is not included in the compromise set, the attacker wins immediately. In SM-security game, a record $(\mathtt{B}, t, i, m, c)$ is added into the compromise set only in either following cases: (1) Party $\mathtt{A}$'s state is corrupted via CORRUPT-A oracle after the ciphertext c was honestly produced via TRANSMIT-B or CHALLENGE-B oracles but before c is delivered to $\mathtt{A}$, or (2) c is produced by $\mathtt{B}$ within $\triangle_{\mathsf{SM}}$ epochs since the last corruption of either party's state. Finally, the experiment updates $\mathtt{A}$'s epoch to the maximum of $t_{\mathtt{A}}$ and t', deletes the record corresponding to the position (t', i') in all sets, and returns (t', i', m').

The simplified authenticity experiment additionally inputs two arguments $(t^\star_{\mathsf{L}}, t^\star)$ with $t^\star \geq t^\star_{\mathsf{L}} + \triangle_{\mathsf{SM}}$ from the attacker, where $t^\star$ is the epoch the attacker is trying to attack and $t^\star_{\mathsf{L}}$ is the last corruption event before the attempt. Consider the case $t^\star$ is even, i.e., when $\mathtt{A}$ is the receiver and $\mathtt{B}$ is the sender:

- if the state of either party is leaked at any epoch in $\{t^\star_{\mathsf{L}} + 1, ..., t^\star - 1\}$, the attacker loses immediately;
- if $\mathtt{A}$ is corrupted any time once $t_{\mathtt{A}} > t^\star_{\mathsf{L}}$, the attacker loses the game immediately;
- the inject oracles are reduced except for ciphertexts corresponding to epoch $t^\star$. By reduced injection oracles, it in particular means that if ciphertext c does not correspond to $(t', i') \in \mathsf{comp}$[4], the oracle INJECT-A immediately returns $(t', i', \perp)$.

The SM construction ACD19 makes use of three components: CKA, FS-AEAD, and PRF-PRNG. Notably, the FS-AEAD is used for deterministically sending messages without receiving a reply, making use of the symmetric state shared by

[4] The SM schemes are assumed to be natural. In particular, this means that the position (t', i') output by eRcv can be efficiently computed from c. See Definition 2.

both parties. In particular, this means that the state corruption of either party enables an attacker to trivially forge messages corresponding to *any* subsequent positions in the same epoch.

Attack: [1, Theorem 1] shows that a SM scheme with simplified correctness, privacy, and in particular, authenticity, must be SM-secure. Below, we show that our attack effectively breaks the SM-security with respect to any parameter $\triangle_{\mathsf{SM}} > 0$ of the $\mathsf{ACD19}$, which is respectively proven simplified correct, private, and authentic with respect to the parameter $\triangle_{\mathsf{SM}}$ in [1, Theorem 7, 8, 12] followed by presenting the flaws in the proof of [1, Theorem 1].

The attacker executes as follows:

1. At epoch 0[5], the attacker queries Corrupt-B to corrupt $\mathtt{B}$'s state $\mathsf{st}^0_{\mathtt{B}}$.
2. The attacker queries Transmit-B$(m_1, \bot)$ and Transmit-B$(m_2, \bot)$ for arbitrary messages m_1 and m_2 and receives ciphertext c_1 and c_2, respectively.
3. The attacker queries Deliver-A(c_2).
4. The attacker queries $c \overset{\$}{\leftarrow}$ Transmit-A$(_, \bot)$ followed by Deliver-B(c) and $c' \overset{\$}{\leftarrow}$ Transmit-B$(_, \bot)$ followed by Deliver-A(c') in turn for sufficient rounds, until the safe injection predicate $\mathsf{safe\text{-}inj}$ becomes true.
5. Arriving at any epoch $t_{\mathtt{A}}$, the attacker encrypts a message $\tilde{m} \neq \bot$ corresponding to epoch 0 and message index 3 using $\mathsf{st}^0_{\mathtt{B}}$ and obtains the ciphertext $\tilde{c}$. If $\tilde{c}$ equals any previous ciphertexts, the attacker simply repeats this step for a new message $\tilde{m} \neq \bot$.
6. The attacker queries Inject-A$(\tilde{c})$.

Note that $\tilde{c}$ does not equal any previous ciphertext in the game, the requirement $(\mathtt{B}, \tilde{c}) \notin \mathsf{trans}$ is always true. Moreover, $\mathsf{safe\text{-}inj}$ is also true in the Inject-A, since $t_{\mathtt{A}}$ is sufficiently apart from the epoch 0. Then, the algorithm $\mathsf{Rcv}(\mathsf{st}_{\mathtt{A}}, \tilde{c})$ is invoked for outputting $(\mathsf{st}'_{\mathtt{A}}, t', i', m')$. Note that $\mathtt{A}$'s state of the $\mathsf{FS\text{-}AEAD}$ component at epoch 0 is not erased from the state, since c_1 has not been delivered[6]. This means, the Rcv algorithm can efficiently recovers $(t', i', m') = (0, 3, \tilde{m})$. Besides, since $\tilde{c}$ is produced by the attacker, instead of making use Transmit-B or Challenge-B oracles, no records matching $(\mathtt{B}, 0, 3)$ is included in the compromise set. Thus, the attacker wins.

Flaws in the proof of [1, Theorem 1]: The proof of [1, Theorem 1] is given by hybrid games. In particular in [1, Lemma 17], they reduce the gap between a game H_1 and a game H_2, which is identical to H_1 but with reduced injection oracles, to the simplified authenticity.

Let $\mathcal{A}$ denote an attacker that can distinguish H_1 and H_2. The reduction $\mathcal{B}$ guesses

– *"the epoch $t^\star$ of the first successful injection query $\mathcal{A}$ makes, and"*

[5] Here we use epoch 0 as example. Similar attacks also work starting at any epoch.

[6] We stress that although the $\mathsf{ACD19}$ includes a so-called $\mathsf{FS\text{-}Max}$ algorithm, which remembers how many messages should be received at epoch 0. This algorithm is only used to erase memory for epoch 0 as long as all messages in epoch 0 have been delivered [1, Section 4.2.1], but not to reject any ciphertext with index out of range.

– *"the time $t^\star_{\mathsf{L}}$ of the last corruption event before the healing event that should have protected against the injection"*

by sampling them uniformly at random but with $t^\star \geq t^\star_{\mathsf{L}} + \triangle_{\mathsf{SM}}$.

However, if $\mathcal{A}$ sends the corruption and injection queries as explained in our attack, which are allowed in the SM-security game, it holds that $t^\star_{\mathsf{L}} = t^\star = 0$. Thus, the reduction can never guess correctly for any $\triangle_{\mathsf{SM}} > 0$. This implies that whether $\mathcal{B}$ can win the simplified authenticity game is *independent* of whether $\mathcal{A}$ can distinguish H_1 and H_2.

Impact of our attack: Note that the SM scheme aims to modularize the double ratchet framework in Signal. It is natural to ask whether Signal protocol itself also suffers from the similar attacks. Fortunately, the answer is: *NO*. Once the party receives the counter, which indicates how many messages were sent in the previous epoch, the receiver immediately pre-computes all symmetric keys for decrypting undelivered ciphertexts that correspond to the previous epoch. When the ciphertext corresponding to a previous epoch arrives in the future, the receiver simply reads the decryption key from the local memory, instead of deriving it using double ratchet. This mechanism effectively prevents our attack.

C.2 Simplified Security Models

Due to the attacks presented above, we have to define some of our simplified security models differently from the ones in [1].

Correctness: We define our correctness model $\mathsf{Exp}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ identical to the model $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ with the same parameter $\triangle_{\mathsf{eSM}}$, except for the following modifications:

1. there are no Challenge-A and Challenge-B oracles
2. the Inject-A and Inject-B are replaced by a reduced injection oracle, which is identical to the injection oracle except for the following two modifications:
   – if the input ciphertext c does not correspond to any position $(t', i') \in \mathsf{comp}$, Inject-A and Inject-B immediately returns $(t', i', \perp)$
   – the if-clause in Line 69 and 70 are removed

This simplified correctness experiment is defined similar to the one in [1].

Note that the attacker receives no information about the challenge bit, since the challenge oracles are removed. The attacker cannot win via the predicate $\mathsf{win}^{\mathsf{priv}}$ except by randomly guessing. Moreover, the predicate $\mathsf{win}^{\mathsf{auth}}$ in the injection oracles is removed. The $\mathsf{win}^{\mathsf{auth}}$ predicate is never set to true. Intuitively, the attacker can win the correctness game with non-zero advantage only via $\mathsf{win}^{\mathsf{corr}}$ in the Deliver-A and Deliver-B oracles.

Definition 15. *An* eSM *scheme Π is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-*CORR *secure if the below defined advantage for any attacker $\mathcal{A}$ in time t is bounded by*

$$\mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \Pr[\mathsf{Exp}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)] \leq \epsilon$$

, where q, q_{ep}, and q_{M} respectively denote the maximal number of queries $\mathcal{A}$ can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment $\mathsf{Exp}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

Authenticity: We define our authenticity model $\mathsf{Exp}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ identical to the model $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ with the same parameter $\triangle_{\mathsf{eSM}}$, except for the following modifications:

1. there are no Challenge-A and Challenge-B oracles
2. the winning predicate $\mathsf{win}^{\mathsf{corr}}$ is never set to true in the Deliver-A and Deliver-B, i.e., the if-clause in Line 61 is removed.
3. the attacker has to output an epoch $t^\star$ at the beginning of the experiment
4. the Inject-A(c) and Inject-B(c) are replaced by a reduced injection oracle (see above) unless the input ciphertext c corresponds to the epoch $t^\star$. (Recall that the position including the epoch and message index is assumed to be efficiently computable from c for natural eSM.)

This simplified authenticity experiment is defined differently from the one in [1], as the attacker has to output only one epoch, which indicates the epoch of the forged ciphertext.

Note that the attacker receives no information about the challenge bit, since the challenge oracles are removed. The attacker cannot win via the predicate $\mathsf{win}^{\mathsf{priv}}$ except by randomly guessing. Moreover, the predicate $\mathsf{win}^{\mathsf{corr}}$ in the deliver oracles is removed. The $\mathsf{win}^{\mathsf{corr}}$ predicate is never set to true. Intuitively, the attacker can win the authenticity game with non-zero advantage only via $\mathsf{win}^{\mathsf{auth}}$ in the Inject-A and Inject-B oracles for a forged ciphertext corresponding to the epoch $t^\star$, which is claimed by the attacker at the beginning of the experiment.

Definition 16. *An* eSM *scheme* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-$\mathsf{AUTH}$ *secure if the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \Pr[\mathsf{Exp}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (\_, 1, \_)] \leq \epsilon$$

, where q, q_{ep}, and q_{M} respectively denote the maximal number of queries $\mathcal{A}$ can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment $\mathsf{Exp}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

Privacy: We define our privacy model $\mathsf{Exp}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$ for an eSM scheme Π with respect to a parameter $\triangle_{\mathsf{eSM}}$ identical to the model $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ with the same parameter $\triangle_{\mathsf{eSM}}$, except for the following modifications:

1. the winning predicate $\mathsf{win}^{\mathsf{corr}}$ is never set to true in the Deliver-A and Deliver-B, i.e., the if-clause in Line 61 is removed.
2. the Inject-A(c) and Inject-B(c) are replaced by a reduced injection oracle (see above).
3. the attacker has to output an epoch $t^\star$ at the beginning of the experiment.
4. the challenge oracle Challenge-A (resp. Challenge-B) can only be queried if $t_{\mathtt{A}} = t^\star$ (resp. $t_{\mathtt{B}} = t^\star$)

This simplified privacy experiment is also defined differently from the one in [1], as the attacker has to output only one epoch, which indicates the epoch of the challenge query.

Note that the predicate $\mathsf{win}^{\mathsf{corr}}$ in the deliver oracles and the $\mathsf{win}^{\mathsf{auth}}$ in the injection oracles are removed. The $\mathsf{win}^{\mathsf{corr}}$ and $\mathsf{win}^{\mathsf{auth}}$ predicates are never set to true. Intuitively, the attacker can win the privacy game only via $\mathsf{win}^{\mathsf{priv}}$ predicate by distinguishing the challenge bit using the challenge ciphertexts corresponding to the epoch $t^\star$, which is claimed by the attacker at the beginning of the experiment.

Definition 17. *An* eSM *scheme* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-PRIV *secure if the below defined advantage for any attacker* $\mathcal{A}$ *in time* t *is bounded by*

$$\mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) := \Pr[\mathsf{Exp}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (\_, \_, 1)] \leq \epsilon$$

, where q*,* q_{ep}*, and* q_{M} *respectively denote the maximal number of queries* $\mathcal{A}$ *can make, the maximal number of epochs, and the maximal number of pre-keys of each party in the experiment* $\mathsf{Exp}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

From Simplified Security to eSM *Security:* It is straightforward that if an eSM scheme is eSM secure, then it simultaneously satisfies the simplified CORR, AUTH, and PRIV security. The following theorem shows the implication also holds in the opposite direction. We give the full proof in Supplementary Material D.1.

Theorem 1 *Let* Π *be an* eSM *scheme that is*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon^{\mathsf{CORR}}_{\Pi})$-CORR *secure,*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon^{\mathsf{AUTH}}_{\Pi})$-AUTH *secure, and*
- $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon^{\mathsf{PRIV}}_{\Pi})$-PRIV *secure*

Then, it is also $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-eSM *secure, where*

$$\epsilon \leq \epsilon^{\mathsf{CORR}}_{\mathsf{eSM}} + q_{\mathsf{ep}}(\epsilon^{\mathsf{AUTH}}_{\mathsf{eSM}} + \epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}})$$

D Proof of Theorems

D.1 Proof of Theorem 1

Proof. The proof is conducted by case distinction. Let $\mathcal{A}$ denote an attacker that breaks $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ security of an eSM scheme Π with respect to the parameter $\triangle_{\mathsf{eSM}}$. Recall that the advantage of $\mathcal{A}$ in winning $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment is defined as:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{eSM}}_{\mathsf{eSM},\triangle_{\mathsf{eSM}}}(\mathcal{A}) = \max\Big(&\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)],\\ &\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)],\\ &|\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|\Big)\end{aligned}$$

Below, we respectively measure $\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)]$, $\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)]$, and $|\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|$ in the following Case 1, 2, and 3.

Case 1. We compute the probability $\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)]$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{corr}}$ by reduction. Namely, if $\mathcal{A}$ can win $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment of the eSM construction Π with a parameter $\triangle_{\mathsf{eSM}}$, then there exists an attacker $\mathcal{B}_1$ that breaks simplified CORR security of the eSM construction Π with the same parameter $\triangle_{\mathsf{eSM}}$. Let $\mathcal{C}_1$ denote the challenger in the $\mathsf{Exp}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. At the beginning, the attacker $\mathcal{B}_1$ samples a challenge bit $\mathsf{b} \in \{0,1\}$ uniformly at random. Then, $\mathcal{B}_1$ invokes $\mathcal{A}$ and answers the queries from $\mathcal{A}$ as follows. Note that all safe predicates in eSM and CORR experiments are identical, $\mathcal{B}_1$ can always compute the safe predicates by itself, according to $\mathcal{A}$'s previous queries.

- NewIdKey-A(r) and NewIdKey-B(r): $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- NewPreKey-A(r) and NewPreKey-B(r): $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- RevIdKey-A and RevIdKey-B: $\mathcal{B}_1$ sets $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{A}}$ or $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{B}}$ (according the invoked oracle) to false and runs **corruption-update**(). For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_1$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_1$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_1$ resets the safe identity predicate to true. Then, the attacker $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- RevPreKey-A and RevPreKey-B: $\mathcal{B}_1$ adds the corresponding pre-key counter into the pre-key reveal list, according to the invoked oracle and runs **corruption-update**(). For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_1$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_1$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_1$ removes the pre-ket counter from the pre-key reveal list. Then, the attacker $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- Corrupt-A and Corrupt-B: Let P denote the party, whose session state the attacker is trying to corrupt. $\mathcal{B}_1$ adds the corresponding epoch counter t_{P} into the session state corruption list $\mathcal{L}^{\mathsf{cor}}_{\mathsf{P}}$ and runs **corruption-update**(). Next, $\mathcal{B}_1$ checks whether there exists a record including $(\neg\mathsf{P}, \mathsf{ind}, \mathsf{flag}) \in \mathsf{chall}$. If such element does not exist, or, such element exists but either of the following conditions holds,
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{idK}}_{\mathsf{P}}$
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{preK}}_{\mathsf{P}}(\mathsf{ind})$

  If one of them is false, $\mathcal{B}_1$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_1$ removes the epoch counter t_{P} from the session state corruption list. Then, the attacker $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- Transmit-A(m,r) and Transmit-B(m,r): $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.
- Challenge-A(m_0,m_1,r) and Challenge-B(m_0,m_1,r): We first consider the case for answering Challenge-A(m_0,m_1,r). The attacker $\mathcal{B}_1$ first com-

putes $\mathsf{flag} = [\![r = \bot]\!]$. Namely, $\mathsf{flag} = \text{true}$ if and only if r is $\bot$. Then, $\mathcal{B}_1$ checks whether the predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}})$ is true, according to $\mathcal{A}$'s previous queries. If the safe predicates is false, or, the input messages m_0 and m_1 have the distinct length, $\mathcal{B}_1$ simply aborts the oracle. Otherwise, $\mathcal{B}_1$ queries Transmit-A(m_{b}, r) to $\mathcal{C}_1$ for a ciphertext c. Then, $\mathcal{B}_1$ adds the record $\mathbf{record}(\mathtt{A}, n_{\mathtt{B}}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, m_{\mathsf{b}}, c)$ into its own $\mathsf{allChall}$ and chall. Finally, $\mathcal{B}_1$ returns c to $\mathcal{A}$.
The step for answering Challenge-B(m_0, m_1, r) is similar to above step except that the functions and variables related to $\mathtt{A}$ are replaced by the ones to $\mathtt{B}$ and vice versa.

– Deliver-A(c) and Deliver-B(c): $\mathcal{B}_1$ first checks whether there exists an element $(t, i, c) \in \mathsf{chall}$ for any t and i. If such element exists, the attacker $\mathcal{B}_1$ simply returns $(t, i, \bot)$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_1$ simply forwards the queries to $\mathcal{C}_1$, followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$. After that, $\mathcal{B}_1$ removes any element including (t, i, c) from the challenge set chall.
– Inject-A(c) and Inject-B(c): $\mathcal{B}_1$ simply forwards them to $\mathcal{C}_1$ followed by forwarding replies from $\mathcal{C}_1$ to $\mathcal{A}$.

Note that if the attacker $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{corr}}$, the winning predicate $\mathsf{win}^{\mathsf{auth}}$ in the Inject-A(c) and Inject-B(c) is never set to true, which implies either $m' = \bot$ or $(\mathtt{B}, t', i') \in \mathsf{comp}$, where t' and i' can be efficiently computed from the input ciphertext c. This means, the reduced injection oracles are identical to the original injection oracles from $\mathcal{A}$'s view. Moreover, all other oracles are honestly simulated. This means, $\mathcal{B}_1$ wins if and only if $\mathcal{A}$ wins. Thus, we have that

$$\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)] \leq \mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{B}_1) \leq \epsilon^{\mathsf{CORR}}_{\Pi}$$

Furthermore, if $\mathcal{A}$ runs in time t, so does $\mathcal{B}_1$.

Case 2. We compute the probability $\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)]$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{auth}}$ by reduction.

Namely, if $\mathcal{A}$ can win $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment of a eSM construction Π with a parameter $\triangle_{\mathsf{eSM}}$, then there exists an attacker $\mathcal{B}_2$ that breaks simplified AUTH security of the eSM construction Π with the same parameter $\triangle_{\mathsf{eSM}}$. Let $\mathcal{C}_2$ denote the challenger in the $\mathsf{Exp}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. At the beginning, the attacker $\mathcal{B}_2$ samples a challenge bit $\mathsf{b} \in \{0,1\}$ and an epoch $t^\star \in [q_{\mathsf{ep}}]$ uniformly at random. Next, $\mathcal{B}_2$ sends $t^\star$ to its challenger $\mathcal{C}_2$. Then, $\mathcal{B}_2$ invokes $\mathcal{A}$ and answers the queries from $\mathcal{A}$ as follows. Note that all safe predicates in eSM and AUTH experiments are identical, $\mathcal{B}_2$ can always compute the safe predicates by itself, according to $\mathcal{A}$'s previous queries.

– NewIdKey-A(r) and NewIdKey-B(r): $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
– NewPreKey-A(r) and NewPreKey-B(r): $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
– RevIdKey-A and RevIdKey-B: $\mathcal{B}_2$ sets $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}$ or $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}$ (according the invoked oracle) to false and runs $\mathbf{corruption\text{-}update}()$. For each record in

the $\mathsf{allChall}$ set, $\mathcal{B}_2$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_2$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_2$ resets the safe identity predicate to true. Then, the attacker $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.

- RevPreKey-A and RevPreKey-B: $\mathcal{B}_2$ adds the corresponding pre-key counter into the pre-key reveal list, according to the invoked oracle and runs **corruption-update**(). For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_2$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_2$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_2$ removes the pre-ket counter from the pre-key reveal list. Then, the attacker $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- Corrupt-A and Corrupt-B: Let $\mathtt{P}$ denote the party, whose session state the attacker is trying to corrupt. $\mathcal{B}_2$ adds the corresponding epoch counter $t_{\mathtt{P}}$ into the session state corruption list $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}}$ and runs **corruption-update**(). Next, $\mathcal{B}_2$ checks whether there exists a record including $(\neg\mathsf{P}, \mathsf{ind}, \mathsf{flag}) \in \mathsf{chall}$. If such element does not exist, or, such element exists but either of the following conditions holds,
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}_{\mathtt{P}}^{\mathsf{idK}}$
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}_{\mathtt{P}}^{\mathsf{preK}}(\mathsf{ind})$

  If one of them is false, $\mathcal{B}_2$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_2$ removes the epoch counter $t_{\mathtt{P}}$ from the session state corruption list. Then, the attacker $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- Transmit-A(m, r) and Transmit-B(m, r): $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.
- Challenge-A(m_0, m_1, r) and Challenge-B(m_0, m_1, r): We first consider the case for answering Challenge-A(m_0, m_1, r). The attacker $\mathcal{B}_2$ first computes $\mathsf{flag} = [\![r = \bot]\!]$. Namely, $\mathsf{flag} = \text{true}$ if and only if r is $\bot$. Then, $\mathcal{B}_2$ checks whether the predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag})$ is true, according to $\mathcal{A}$'s previous queries. If the safe predicates is false, or, the input messages m_0 and m_1 have the distinct length, $\mathcal{B}_2$ simply aborts the oracle. Otherwise, $\mathcal{B}_2$ queries Transmit-A(m_{b}, r) to $\mathcal{C}_2$ for a ciphertext c. Then, $\mathcal{B}_2$ adds the record **record**$(\mathtt{A}, n_{\mathtt{B}}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, m_{\mathsf{b}}, c)$ into its own $\mathsf{allChall}$ and chall. Finally, $\mathcal{B}_2$ returns c to $\mathcal{A}$.

  The step for answering Challenge-B(m_0, m_1, r) is similar to above step except that the functions and variables related to $\mathtt{A}$ are replaced by the ones to $\mathtt{B}$ and vice versa.
- Deliver-A(c) and Deliver-B(c): $\mathcal{B}_2$ first checks whether there exists an element $(t, i, c) \in \mathsf{chall}$ for any t and i. If such element exists, the attacker $\mathcal{B}_2$ simply returns $(t, i, \bot)$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_2$ simply forwards the queries to $\mathcal{C}_2$, followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$. After that, $\mathcal{B}_2$ removes any element including (t, i, c) from the challenge set chall.

- INJECT-A(c) and INJECT-B(c): $\mathcal{B}_2$ simply forwards them to $\mathcal{C}_2$ followed by forwarding replies from $\mathcal{C}_2$ to $\mathcal{A}$.

Note that if the attacker $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{auth}}$, the winning predicate $\mathsf{win}^{\mathsf{corr}}$ in the DELIVER-A(c) and DELIVER-B(c) is never set to true. This means, the deliver oracles in CORR experiment is identical to the original deliver oracles from $\mathcal{A}$'s view. Note also that the winning predicate $\mathsf{win}^{\mathsf{auth}}$ is never set to false once it has been set to true.

Assume that attacker $\mathcal{B}_2$ guesses the epoch $t^\star$ correctly, such that $\mathcal{A}$ triggers the flip of $\mathsf{win}^{\mathsf{auth}}$ by querying INJECT-A(c) or INJECT-B(c) for a ciphertext c corresponding to epoch $t^\star$, which happens with probability $\frac{1}{q_{\mathsf{ep}}}$. For all previous queries INJECT-A(c) and INJECT-B(c), where c does not correspond to the epoch $t^\star$, the flip of $\mathsf{win}^{\mathsf{auth}}$ from false to true will not be triggered. In this case, our reduced injection oracle correctly simulates the behavior of the original injection oracles. For all previous queries INJECT-A(c) and INJECT-B(c), where c corresponds to the epoch $t^\star$, our reduced injection oracle simulates the identical behavior of the original injection oracles.

Note that all other oracles are honestly simulated. The attacker $\mathcal{B}_2$ wins if and only if $\mathcal{A}$ wins and the guess $t^\star$ is correctly. Note also that the event $\mathcal{A}$ wins and the number that $\mathcal{B}_2$ guesses are independent. Thus, we have that

$$\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)] \le q_{\mathsf{ep}}\mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}} \le q_{\mathsf{ep}}\epsilon^{\mathsf{AUTH}}_{\Pi}$$

Moreover, if $\mathcal{A}$ runs in time t, so does $\mathcal{B}_2$.

Case 3. We compute the probability $|\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{priv}}$ by hybrid games. Let G_i denote the simulation of **Game i**.

Game 0. This game is identical to the $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. Thus, we have that

$$\Pr[\mathsf{G}_0(\mathcal{A}) = (0,0,1)] = \Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)]$$

Game i **(**$1 \le i \le q_{\mathsf{ep}}$**).** This game is identical to **Game** $(i-1)$ except the following modifications:

- When the attacker queries CHALLENGE-A(m_0, m_1, r) at epoch i, the challenger first checks whether $|m_0| = |m_1|$ and aborts if the condition does not hold. Then, the challenger samples a random message $\bar{m}$ of the length $|m_0|$ and runs CHALLENGE-A($\bar{m}, \bar{m}, r$) instead of CHALLENGE-A(m_0, m_1, r). Finally, the challenger returns the produced ciphertext c to $\mathcal{A}$.

It is easy to observe that in **Game** q_{ep} all challenge ciphertexts are encrypted independent of the challenge bit. Thus, the attacker $\mathcal{A}$ can output the bit b' only by randomly guessing, which indicates that

$$\Pr[\mathsf{G}_{q_{\mathsf{ep}}}(\mathcal{A}) = (0,0,1)] = \frac{1}{2}$$

Let E denote the event that the attacker can distinguish any two adjacent hybrid games. We have that

$$|\Pr[\mathsf{G}_{i-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_i(\mathcal{A}) = (0,0,1)]| \le \Pr[E]$$

Moreover, note that the modifications in every hybrid game i is independent of the behavior in hybrid game $(i-1)$. Thus, we have that

$$\begin{aligned}
&|\Pr[\mathsf{G}_0(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_{q_{\mathsf{ep}}}(\mathcal{A}) = (0,0,1)]| \\
\leq & |\sum_{i=1}^{q_{\mathsf{ep}}} \Pr[\mathsf{G}_{i-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_i(\mathcal{A}) = (0,0,1)]| \\
\leq & \sum_{i=1}^{q_{\mathsf{ep}}} |\Pr[\mathsf{G}_{i-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_i(\mathcal{A}) = (0,0,1)]| \\
\leq & q_{\mathsf{ep}} \Pr[E]
\end{aligned}$$

Below, we analyze the probability of the occurrence of the event E by reduction. Namely, if $\mathcal{A}$ can distinguish any two adjacent games **Game** $(i-1)$ and **Game** i, then there exists an attacker $\mathcal{B}_3$ that breaks simplified PRIV security of the eSM construction Π with the same parameter $\triangle_{\mathsf{eSM}}$. Let $\mathcal{C}_3$ denote the challenger in the $\mathsf{Exp}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$ experiment. At the beginning, the attacker $\mathcal{B}_3$ sends the epoch i to its challenger $\mathcal{C}_3$ and samples a bit $\bar{\mathsf{b}} \in \{0,1\}$ uniformly at random. Then, $\mathcal{B}_3$ invokes $\mathcal{A}$ and answers the queries from $\mathcal{A}$ as follows. Note that all safe predicates in **Game** $(i-1)$, **Game** i, and PRIV experiments are identical, $\mathcal{B}_3$ can always compute the safe predicates by itself, according to $\mathcal{A}$'s previous queries.

- NewIdKey-A(r) and NewIdKey-B(r): $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- NewPreKey-A(r) and NewPreKey-B(r): $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- RevIdKey-A and RevIdKey-B: $\mathcal{B}_3$ sets $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}$ or $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}}$ (according the invoked oracle) to false and runs **corruption-update**(). For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_3$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_3$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_3$ resets the safe identity predicate to true. Then, the attacker $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- RevPreKey-A and RevPreKey-B: $\mathcal{B}_3$ adds the corresponding pre-key counter into the pre-key reveal list, according to the invoked oracle and runs **corruption-update**(). For each record in the $\mathsf{allChall}$ set, $\mathcal{B}_3$ then checks whether the safe challenge predicate for all of the records holds. If one of them is false, $\mathcal{B}_3$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_3$ removes the pre-ket counter from the pre-key reveal list. Then, the attacker $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.
- Corrupt-A and Corrupt-B: Let $\mathtt{P}$ denote the party, whose session state the attacker is trying to corrupt. $\mathcal{B}_3$ adds the corresponding epoch counter $t_{\mathtt{P}}$ into the session state corruption list $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}}$ and runs **corruption-update**(). Next, $\mathcal{B}_3$ checks whether there exists a record including $(\neg\mathtt{P}, \mathsf{ind}, \mathsf{flag}) \in \mathsf{chall}$. If such element does not exist, or, such element exists but either of the following conditions holds,

- $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}_{\mathtt{P}}^{\mathsf{idK}}$
- $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}_{\mathtt{P}}^{\mathsf{preK}}(\mathsf{ind})$

If one of them is false, $\mathcal{B}_3$ undoes the actions in this query and exists the oracle invocation. In particular, $\mathcal{B}_3$ removes the epoch counter $t_{\mathtt{P}}$ from the session state corruption list. Then, the attacker $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.

– Transmit-A(m, r) and Transmit-B(m, r): $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.

– Challenge-A(m_0, m_1, r) and Challenge-B(m_0, m_1, r): These oracles are answered according to one of the following cases. Here, we only explain the behavior for answering Challenge-A for simplicity. The behavior for answering Challenge-B can be defined analogously.

- $[t_{\mathtt{A}} < i]$: When the attacker $\mathcal{A}$ queries Challenge-A(m_0, m_1, r) at epoch $t_{\mathtt{A}} < i$, the $\mathcal{B}_3$ first computes $\mathsf{flag} \leftarrow [\![r = \bot]\!]$. Next, $\mathcal{B}_3$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}}) = \text{true}$ and $|m_0| = |m_1|$ and aborts if either condition does not hold. Otherwise, $\mathcal{B}_3$ samples a random message $\bar{m}$ of the length $|m_0|$ and queries Transmit-A$(\bar{m}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\mathsf{rec} = (\mathtt{A}, n_{\mathtt{B}}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, \bar{m}, c)$ into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
- $[t_{\mathtt{A}} = i]$: When the attacker $\mathcal{A}$ queries Challenge-A(m_0, m_1, r) at epoch $t_{\mathtt{A}} = i$, the $\mathcal{B}_3$ first computes $\mathsf{flag} \leftarrow [\![r = \bot]\!]$. Next, $\mathcal{B}_3$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}}) = \text{true}$ and $|m_0| = |m_1|$ and aborts if either condition does not hold. Otherwise, $\mathcal{B}_3$ samples a random message $\bar{m}$ of the length $|m_0|$ and queries Challenge-A$(m_{\bar{\mathsf{b}}}, \bar{m}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\mathsf{rec} = (\mathtt{A}, n_{\mathtt{B}}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, \_, c)$ into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
- $[t_{\mathtt{A}} > i]$: When the attacker $\mathcal{A}$ queries Challenge-A(m_0, m_1, r) at epoch $t_{\mathtt{A}} > i$, the $\mathcal{B}_3$ first computes $\mathsf{flag} \leftarrow [\![r = \bot]\!]$. Next, $\mathcal{B}_3$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}}) = \text{true}$ and $|m_0| = |m_1|$ and aborts if either condition does not hold. Otherwise, $\mathcal{B}_3$ queries Transmit-A$(m_{\bar{\mathsf{b}}}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\mathsf{rec} = (\mathtt{A}, n_{\mathtt{B}}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, m_{\bar{\mathsf{b}}}, c)$ into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.

– Deliver-A(c) and Deliver-B(c): $\mathcal{B}_3$ first checks whether there exists an element $(t, i, c) \in \mathsf{chall}$ for any t and i. If such element exists, the attacker $\mathcal{B}_3$ simply returns $(t, i, \bot)$ to $\mathcal{A}$. Otherwise, $\mathcal{B}_3$ simply forwards the queries to $\mathcal{C}_3$, followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$. After that, $\mathcal{B}_2$ removes any element including (t, i, c) from the challenge set chall.

– Inject-A(c) and Inject-B(c): $\mathcal{B}_3$ simply forwards them to $\mathcal{C}_3$ followed by forwarding replies from $\mathcal{C}_3$ to $\mathcal{A}$.

Note that if the attacker $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{priv}}$, the winning predicate $\mathsf{win}^{\mathsf{corr}}$ in the Deliver-A(c) and Deliver-B(c) and $\mathsf{win}^{\mathsf{auth}}$ in the Inject-A(c) and Inject-B(c) is never set to true. This means, the deliver oracles and injection oracles in PRIV experiment is identical to the original ones from $\mathcal{A}$'s view.

Note that all other oracles are honestly simulated. If the challenge bit b in the PRIV experiment is 0, then $\mathcal{B}_3$ perfectly simulates **Game** $(i-1)$ to $\mathcal{A}$. If the challenge bit b in the PRIV experiment is 1, then $\mathcal{B}_3$ perfectly simulates **Game** i to $\mathcal{A}$. This means, the attacker $\mathcal{B}_3$ wins if and only if $\mathcal{A}$ can distinguish the adjacent hybrid games **Game** $(i-1)$ and **Game** i, which is defined as the occurrence of event E. Thus, we have that

$$\Pr[E] \leq \mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}} \leq \epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}}$$

Combing the equations above, we have that:

$$\begin{aligned}
&|\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}| \\
=&|\Pr[\mathsf{G}_0(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_{q_{\mathsf{ep}}}(\mathcal{A})]| \\
\leq& q_{\mathsf{ep}} \Pr[E] \leq q_{\mathsf{ep}} \epsilon^{\mathsf{PRIV}}_{\mathsf{eSM}}
\end{aligned}$$

Moreover, if $\mathcal{A}$ runs in time t, so does $\mathcal{B}_2$.

Conclusion. The proof is concluded by

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{eSM}}_{\mathsf{eSM},\triangle_{\mathsf{eSM}}}(\mathcal{A}) =& \max\Big(\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (1,0,0)], \\
& \Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,1,0)], \\
& |\Pr[\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|\Big) \\
\leq& \max\left(\epsilon^{\mathsf{CORR}}_{\Pi}, q_{\mathsf{ep}}\epsilon^{\mathsf{AUTH}}_{\Pi}, q_{\mathsf{ep}}\epsilon^{\mathsf{PRIV}}_{\Pi}\right) \\
\leq& \epsilon^{\mathsf{CORR}}_{\Pi} + q_{\mathsf{ep}}(\epsilon^{\mathsf{AUTH}}_{\Pi} + \epsilon^{\mathsf{PRIV}}_{\Pi})
\end{aligned}$$

D.2 Proof of Theorem 2

Proof. The proof is given by a sequence of games. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning **Game** i.
Game 0. This game is identical to the $\mathsf{Exp}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, we have that

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}}$$

Game 1. In this game, if the attacker queries INJECT-A(ind, c) and INJECT-B(ind, c) with c corresponding to position $(t^\star, i^\star)$ such that $t^\star \leq \min(t_{\mathtt{A}}, t_{\mathtt{B}}) - 2$, the challenger immediately returns $(t^\star, i^\star, \perp)$.

Note that the oracles are fined symmetric for party $\mathtt{A}$ and $\mathtt{B}$. Without the loss of generality, we only explain the case for INJECT-A(ind, c) and $t^\star$ is even. The case for INJECT-B and t is odd can be given analogously.

In fact, recall that the eRcv algorithm is executed in INJECT-A(ind, c) oracle only if the following conditions hold

1. $(\mathtt{B}, c) \notin \mathsf{trans}$
2. $\mathsf{ind} \leq n_{\mathtt{A}}$
3. $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{B}}) = \text{true}$ **and** $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{A}}) = \text{true}$ which are equivalent to $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}}) = \text{true}$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{A}}) = \text{true}$
4. $(t^\star, i^\star) \in \mathsf{comp}$, where $(t^\star, i^\star)$ is the position of the input ciphertext c

Recall that $(t^\star, i^\star) \in \mathsf{comp}$ means that a ciphertext at this position has been produced by a party, which implies that $t^\star \leq \max(t_{\mathtt{A}}, t_{\mathtt{B}})$. Moreover, a ciphertext is added into comp only when

1. in the CORRUPT-A oracle, if $\mathsf{safe\text{-}st}(t^\star) = \text{false}$ holds.
2. in the CORRUPT-B oracle at epoch $t_{\mathtt{B}} = t^\star$, which means $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$
3. in the TRANSMIT-B oracle, if $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t^\star) = \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$ holds
4. in the REVIDKEY-A, REVIDKEY-B, REVPREKEY-A, REVPREKEY-B oracles, if $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t^\star) = \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$

In all of the above cases, we know that $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$. Note that the conditions $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}}) = \text{false}$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{A}}) = \text{false}$ must hold at the same time. This means, $t^\star \leq \min(t_{\mathtt{A}}, t_{\mathtt{B}}) - 2$. Thus, **Game 0** and **Game 1** are identical from the attacker's view. Thus, we have that

$$\mathsf{Adv}_0 = \mathsf{Adv}_1$$

In particular, this also means that both parties have already received at least one message in the epoch $t^\star$ and have produced the root keys before the INJECT-A and INJECT-B for ciphertexts corresponding $t^\star$ are queried.
Game 2. This game is identical to **Game 1** except the following modification:

1. Whenever the challenger executes TRANSMIT-A and TRANSMIT-B to enter a new epoch $t^\star$, the challenger records the root key $rk' \leftarrow \mathsf{st}.rk$ produced during the oracle. When DELIVER-A or DELIVER-B is invoked on the first ciphertext that corresponds to the epoch $t^\star$, the challenger replaces the derivation of the root key rk by the recorded rk'.

The gap between **Game 1** and **Game 2** can be analyzed by a sequence of hybrid games, where each hybrid only replace the root key at one epoch. Note that if the receiver executes the eRcv algorithm for the first message in a new epoch. The new $\mathsf{st}.rk$ is derived only when the output of $\mathsf{D.Vrfy}$ is true, which happens except probability δ_{DS}. Note also that the DELIVER-A and DELIVER-B oracles are used to simulate the transmission of the original data that were produced. The honest KEM ciphertexts are delivered to the receiver and will be decrypted using the corresponding private keys. All of them are correctly recovered except probability at most $3\delta_{\mathsf{KEM}}$. If both parties' local root keys are identical, which is true due to the previous hybrid game, the root keys of both parties in this epoch are also identical in this hybrid game. Note that there are at most q_{ep} epochs. Thus, we have that

$$\mathsf{Adv}_1 \le \mathsf{Adv}_2 + q_{\mathsf{ep}}(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}})$$

Game 3. This game is identical to **Game 2** except the following modification:

1. Whenever the challenger executes TRANSMIT-A and TRANSMIT-B, the challenger records the message key $mk' \leftarrow mk$ produced during the oracle together with the position. When DELIVER-A or DELIVER-B is invoked on a ciphertext, the challenger searches the mk at the location of the input c, followed by replacing the derivation of the message key mk by the recorded mk'.

This game is similar to **Game 2**. The only difference is that the challenger runs q hybrid games but not q_{ep}, where q denotes the maximal queries that $\mathcal{A}$ can make. Thus, we can easily have that

$$\mathsf{Adv}_2 \le \mathsf{Adv}_3 + q(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}})$$

Game 4. This game s identical to **Game 3** except the following modification:

1. Whenever the challenger executes TRANSMIT-A(m, r) and TRANSMIT-B(m, r), the challenger records the message m produced during the oracle together with the position. When DELIVER-A or DELIVER-B is invoked on a ciphertext, the challenger searches the message m' at the location of the input c, followed by replacing the recovery of the message m by the recorded m'.

This game is similar to **Game 3**. The only difference is that the challenger runs q hybrid games on the scheme SKE which is deterministic and δ_{SKE}-correct. Similarly, we can easily have that

$$\mathsf{Adv}_3 \le \mathsf{Adv}_4 + q\delta_{\mathsf{SKE}}$$

Final Analysis of Game 4: Now, whenever DELIVER-A or DELIVER-B is delivered, the original messages are always correctly recovered and output with the correct position, which means the attacker never wins. Thus, we have that

$$\mathsf{Adv}_5 = 0$$

The following equation concludes the proof.

$$\begin{aligned}\mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}} &\le q_{\mathsf{ep}}(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}}) + q(\delta_{\mathsf{DS}} + 3\delta_{\mathsf{KEM}} + \delta_{\mathsf{SKE}}) \\ &= (q_{\mathsf{ep}} + q)\delta_{\mathsf{DS}} + 3(q_{\mathsf{ep}} + q)\delta_{\mathsf{KEM}} + q\delta_{\mathsf{SKE}}\end{aligned}$$

D.3 Proof of Theorem 3

Proof. The proof is given by a sequence of games. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning Game i. At the beginning of the experiment, the attacker $\mathcal{A}$ outputs a target epoch $t^\star$, such that it only queries challenge oracles in this epoch. Without loss of generality, we assume $t^\star$ is odd, i.e., $\mathtt{A}$ is the message sender. The case for $t^\star$ is even can be given analogously.
Game 0. This game is identical to the $\mathsf{Exp}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, we have that

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$$

Game 1. This game is identical to **Game 0** except the following modifications:
1. At the beginning of the game, in addition to the target epoch $t^\star$, the attacker has to output a target message index $i^\star$.
2. The challenge oracle CHALLENGE-A can only be queried for encrypting $i^\star$-th message (i.e., $i_{\mathtt{A}} = i^\star - 1$ before the query and $i_{\mathtt{A}} = i^\star$ after the query) in $t_{\mathtt{A}} = t^\star$.

We analyze the gap between **Game 0** and **Game 1** by hybrid games. Note that $\mathcal{A}$ can query oracles at most q times. There are at most q messages can be encrypted in the target epoch.

Game 1.0. This game is identical to **Game 0**. Thus, we have that

$$\mathsf{Adv}_{1.0} = \mathsf{Adv}_0$$

Game $1.j$**,** $1 \le j \le q$. This game is identical to **Game** $1.(j-1)$ except the following modification:
1. If $\mathcal{A}$ sends challenge oracle CHALLENGE-A(m_0, m_1, r) for encrypting j-th message. The challenger first checks whether m_0 and m_1 have the same length and aborts if the condition does not hold. Then, the challenge samples a random message $\bar{m}$ of the length m_0 and runs CHALLENGE-A$(\bar{m}, \bar{m}, r)$ instead of CHALLENGE-A(m_0, m_1, r). Finally, the challenger returns the produced ciphertext c to $\mathcal{A}$.

It is easy to observe that all challenge ciphertexts are encrypted independent of the challenge bit in **Game** $1.q$. Thus, the attacker can guess the challenge bit only by randomly guessing in **Game** $1.q$, which implies that

$$\mathsf{Adv}_{1.q} = 0$$

Let E denote the event that the attacker $\mathcal{A}$ can distinguish any two adjacent hybrid games. Note that the modification in every hybrid game j is independent of the behavior in hybrid game $(j-1)$. Thus, we have that

$$\mathsf{Adv}_{1.0} = \mathsf{Adv}_{1.0} - \mathsf{Adv}_{1.q} \le q \Pr[E]$$

We compute the probability of the occurrence of the event E by reduction. If $\mathcal{A}$ can distinguish any **Game** $1.(j-1)$ and **Game** $1.j$, then we can construct an attacker $\mathcal{B}_1$ that breaks **Game 1**. The attacker $\mathcal{B}_1$ is executed as follows:

1. When $\mathcal{A}$ outputs an epoch $t^\star$, $\mathcal{B}$ outputs $(t^\star, j)$. Meanwhile, $\mathcal{B}_1$ samples a random bit $\bar{\mathsf{b}} \in \{0,1\}$ uniformly at random.
2. When $\mathcal{A}$ queries Challenge-A, $\mathcal{B}$ answers according one of the following case:
   - $[i_{\mathtt{A}} < j-1]$: When the attacker queries Challenge-A(m_0, m_1, r) when $i_{\mathtt{A}} < j-1$, i.e., for encrypting messages before j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \bot]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}})$ holds and whether m_0 and m_1 have the same length. If either condition does not hold, $\mathcal{B}_1$ simply aborts. Otherwise, $\mathcal{B}_1$ samples a random message $\bar{m}$ of the length m_0 and queries Transmit-A$(\bar{m}, r)$ for a ciphertext c. Finally, $\mathcal{B}_1$ adds the corresponding record into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
   - $[i_{\mathtt{A}} = j-1]$: When the attacker queries Challenge-A(m_0, m_1, r) when $i_{\mathtt{A}} = j-1$, i.e., for encrypting j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \bot]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}})$ holds and whether m_0 and m_1 have the same length. If either condition does not hold, $\mathcal{B}_1$ simply aborts. Otherwise, $\mathcal{B}_1$ samples a random message $\bar{m}$ of the length m_0 and queries Challenge-A$(m_{\bar{\mathsf{b}}}, \bar{m}, r)$ for a ciphertext c. Finally, $\mathcal{B}_1$ adds the corresponding record into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
   - $[i_{\mathtt{A}} > j-1]$: When the attacker queries Challenge-A(m_0, m_1, r) when $i_{\mathtt{A}} > j-1$, i.e., for encrypting messages after j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \bot]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, n_{\mathtt{B}})$ holds and whether m_0 and m_1 have the same length. If either condition does not hold, $\mathcal{B}_1$ simply aborts. Otherwise, $\mathcal{B}_1$ queries Transmit-A$(m_{\bar{\mathsf{b}}}, r)$ for a ciphertext c. Finally, $\mathcal{B}_1$ adds the corresponding record into both $\mathsf{allChall}$ and chall, followed by returning the ciphertext c to $\mathcal{A}$.
3. To answer all other oracles, $\mathcal{B}_1$ first checks whether the safe predicate requirements in individual oracles hold. If so, $\mathcal{B}_1$ simply forward the queries to challenger and returns the reply to $\mathcal{A}$. If not, $\mathcal{B}_1$ simply aborts.

Note that all other oracles are honestly simulated except for Challenge-A. If the challenge bit b in **Game 1** is 0, then $\mathcal{B}_1$ perfectly simulates **Game** 1.$(j-1)$ to $\mathcal{A}$. If the challenge bit b in **Game 1** is 1, then $\mathcal{B}_1$ perfectly simulates **Game** 1.j to $\mathcal{A}$. Thus, if $\mathcal{A}$ can distinguish any adjacent two hybrid games, $\mathcal{B}_1$ wins **Game 1**, which implies $\Pr[E] \leq \mathsf{Adv}_1$, and further

$$\mathsf{Adv}_0 = \mathsf{Adv}_{1.0} \leq q \Pr[E] \leq q\mathsf{Adv}_1$$

Game 2. Let $\mathsf{ind}^\star$ denote the index of $\mathit{prepk}_{\mathtt{B}}$ that is used to encrypt $i^\star$'s message in epoch $t^\star$. Let $\mathsf{flag}^\star$ denote the random quality in the target challenge oracle. In this game, $\mathcal{A}$ wins immediately, if at the end of experiment $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \text{false}$.

Note that before the challenge query, the safe predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t^\star, \mathsf{ind}^\star)$ must hold, i.e.,

$$\Big(\mathsf{safe\text{-}st}_{\mathtt{A}}(t^\star) \textbf{ and } \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big) \textbf{ or } \Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big) \textbf{ or }$$
$$\Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) \textbf{ or } \Big(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big)$$

This means, at least one of the following conditions must hold at the time of query of Challenge-A.

1. $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$
2. $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \text{true}$
3. $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \text{true}$

When querying identity keys or pre-keys oracles, the oracle aborts if it will triggers the safe challenge predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t^\star, \mathsf{ind}^\star)$ to false. When querying corruption oracles, the violation of $\mathsf{safe\text{-}st}_{\mathtt{B}}$ must indicate $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big)$ or $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big)$. Thus, at least one of the above conditions must hold even at the end of experiment

This means, $\mathcal{A}$ cannot gain any additional advantage in winning **Game 2**, which implies that

$$\mathsf{Adv}_1 = \mathsf{Adv}_2$$

Below, we analyze the advantage Adv_2 into three cases, whether $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \text{true}$ or $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\Big) = \text{true}$ or $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$ holds at the end of the experiment.

Case 1: $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \text{true}$.

In this case, $\Big(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathtt{B}}^{\mathsf{idK}}\Big) = \text{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle Challenge-A query. We use Adv_j^{C1} to denote $\mathcal{A}$'s advantage in winning **Game** j in this case. In the remaining of this case analysis, we focus on the epoch $t^\star$ and the message index $i^\star$.

Game C1.3. This game is identical to **Game 2** except the following modification:

1. The challenger additionally samples a random key $k' \in \mathcal{K}$, where $\mathcal{K}$ denote the key space of the underlying KEM.
2. $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$ in Line 16 in Figure 3 is replaced by $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k', k_3)$
3. $k_2 \leftarrow \mathsf{K.Dec}(ik, c_2)$ in Line 32 in Figure 3 is replaced by $k_2 \leftarrow k'$

If $\mathcal{A}$ can distinguish **Game 2** and **Game C1.3**, then we can construct an attacker $\mathcal{B}_2$ that breaks $\mathsf{IND\text{-}CCA}$ security of underlying KEM. The attacker $\mathcal{B}_2$ receives a public key pk, a challenge ciphertext $c^\star$, and a key $k^\star$, and simulates the game as follows:

1. $\mathcal{A}$ outputs $(t^\star, i^\star)$ at the beginning of the game.
2. When $\mathcal{A}$ queries NewIdKey-B(r), checks whether $r = \perp$. If $r \neq \perp$, then $\mathcal{B}_2$ returns pk to $\mathcal{A}$.

3. When $\mathcal{A}$ queries Challenge-A(m_0, m_1, r) for encrypting $i^\star$'s message in the epoch $t^\star$, $\mathcal{B}_2$ aborts if $r \neq \perp$. Then, $\mathcal{B}_2$ honestly runs Challenge-A except replacing $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$ in Line 16 in Figure 3 by $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k^\star, k_3)$
4. When $\mathcal{A}$ queries Deliver-B(c) oracle, where c is output by Challenge-A oracles, $\mathcal{B}_2$ honestly runs the eRcv algorithm except directly using $k^\star$ at the place of k_2 instead of running decapsulation algorithm.
5. When $\mathcal{A}$ queries Inject-B(c) oracle for a ciphertext c, $\mathcal{B}_2$ forwards c to its decapsulation oracle for a key k, followed by use this key in the place of the decapsulated k_2 to run eRcv algorithm.
6. All other oracles are honestly simulated.

Note that if the challenge bit in the $\mathsf{IND\text{-}CCA}$ security experiment equals 0, then $\mathcal{B}_2$ simulates **Game 2** to $\mathcal{A}$. If the challenge bit in the $\mathsf{IND\text{-}CCA}$ security experiment equals 1, then $\mathcal{B}_2$ simulates **Game C1.3** to $\mathcal{A}$. $\mathcal{B}_2$ wins if and only if $\mathcal{A}$ can distinguish **Game 2** and **Game C1.3**. Thus, we have that

$$\mathsf{Adv}_2^{C1} \leq \mathsf{Adv}_3^{C1} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$$

Game C1.4. This game is identical to **Game C1.3** except the following modifications:

1. The challenger additionally samples a random update value $\widetilde{\mathsf{upd}}^{\mathsf{ur}} \in \{0,1\}^\lambda$
2. $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})$ in Line 23 and 39 in Figure 3 is replaced by $mk \leftarrow \mathsf{KDF}_5(urk, \widetilde{\mathsf{upd}}^{\mathsf{ur}})$

If $\mathcal{A}$ can distinguish **Game C1.3** and **Game C1.4**, then we can construct an attacker $\mathcal{B}_3$ that breaks $\mathsf{3prf}$ security of underlying KDF_1. Note that the random key k' is sampled random in **Game C1.3**. $\mathcal{B}_3$ can easily query k_1, k_3 to its oracle on the second input, and use the reply in the place of $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}})$. If the oracle simulates KDF_1, then $\mathcal{B}_3$ simulates **Game C1.3** to $\mathcal{A}$. If the oracle simulates a random function, then $\mathcal{B}_3$ simulates **Game C1.4**. Thus, we have that

$$\mathsf{Adv}_3^{C1} \leq \mathsf{Adv}_4^{C1} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$$

Game C1.5. This game is identical to **Game C1.4** except the following modifications:

1. The challenger additionally samples a random message key $\widetilde{mk} \in \{0,1\}^\lambda$
2. $c' \leftarrow \mathsf{S.Enc}(mk, m)$ in Line 23 and 39 in Figure 3 is replaced by $c' \leftarrow \mathsf{S.Enc}(\widetilde{mk}, m)$

Similar to the game above, if $\mathcal{A}$ can distinguish **Game C1.4** and **Game C1.5**, then we can construct an attacker $\mathcal{B}_4$ that breaks swap security of underlying KDF_5. Note that the random update value $\widetilde{\mathsf{upd}}^{\mathsf{ur}}$ is sampled random in **Game C1.4**. $\mathcal{B}_4$ can easily query urk to its oracle and use the reply in the place of mk. If the oracle simulates KDF_5, then $\mathcal{B}_4$ simulates **Game C1.3** to $\mathcal{A}$. If the oracle simulates a random function, then $\mathcal{B}_3$ simulates **Game C1.5**. Thus, we have that

$$\mathsf{Adv}_4^{C1} \leq \mathsf{Adv}_5^{C1} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}$$

Game Final Analysis for Case 1: In the end, we compute $\mathcal{A}$'s advantage in winning **Game C1.5** by reduction. If $\mathcal{A}$ can win **Game C1.5**, then we can construct an attacker $\mathcal{B}_5$ that breaks $\mathsf{IND\text{-}1CCA}$ security of the underlying SKE. The reduction is simulated as follows:

1. $\mathcal{A}$ outputs $(t^\star, i^\star)$ at the beginning of the game.
2. $\mathcal{B}$ samples a random bit $\bar{\mathsf{b}} \xleftarrow{\$} \{0,1\}$.
3. When $\mathcal{A}$ queries Challenge-A(m_0, m_1, r) for encrypting $i^\star$'s message in the epoch $t^\star$, $\mathcal{B}_5$ aborts if $r \neq \bot$ or m_0 and m_1 have different length. Next, $\mathcal{B}_5$ samples a random message $\bar{m}$ of length $|m_0|$.Then, $\mathcal{B}_5$ queries its challenger on $(\bar{m}, m_{\bar{\mathsf{b}}})$ and receives a ciphertext $c^\star$. After that, $\mathcal{B}_5$ honestly runs Challenge-A except replacing $c' \leftarrow \mathsf{S.Enc}(mk, m)$ in Line 23 and 39 in Figure 3 by $c' \leftarrow c^\star$.
4. When $\mathcal{A}$ queries Deliver-B(c) oracle such that c includes $t^\star$, $i^\star$, and $c^\star$, $\mathcal{B}_5$ honestly simulates Deliver-B except for outputting $m' = \bot$.
5. When $\mathcal{A}$ queries Inject-B(c) oracle for a ciphertext corresponds to the position $(t^\star, i^\star)$, $\mathcal{B}_5$ forwards c to its decapsulation oracle for a message m', followed by outputting $(t^\star, i^\star, m')$
6. All other oracles are honestly simulated.

Note that if the forgery via Inject-B is accepted, then the attacker cannot win via $\mathsf{win}^{\mathsf{priv}}$ predicate since a natural eSM scheme does not accept two messages at the same position. So, $\mathcal{B}_5$ perfectly simulate **Game C1.5** to $\mathcal{A}$ and wins if and only if $\mathcal{A}$ wins. Thus, we have that

$$\mathsf{Adv}_5^{C1} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$$

To sum up, we have that

$$\mathsf{Adv}_2^{C1} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$$

Case 2: $\left(\mathsf{flag}^\star = \mathsf{good} \text{ and } \mathsf{safe}_{\mathsf{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\right) = $ true.

In this case, $\left(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathsf{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\right) = \text{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle Challenge-A query. We use Adv_j^{C2} to denote $\mathcal{A}$'s advantage in winning **Game** j in this case. In the remaining of this case analysis, we focus on the epoch $t^\star$ and the message index $i^\star$.

Game C2.3 In this game, the challenger guesses the index of the pre-key $\mathsf{ind}^\star$ by randomly guessing at the beginning of the experiment. If the guess is wrong, the challenger aborts and let $\mathcal{A}$ immediately win. Note that there are at most q_M in the experiment, the challenger can guess correctly with probability $\frac{1}{q_\mathsf{M}}$. Thus, we have that

$$\mathsf{Adv}_2^{C2} \leq q_\mathsf{M} \mathsf{Adv}_3^{C2}$$

Game C2.4, C2.5, C2.6. These games are defined similar to **Game C1.3, C1.4, C1.5**. The only difference is to apply the modification not to B's identity key but B's $\mathsf{ind}^\star$-th pre-key. The proof can be easily given in a similar way and we have that

$$\mathsf{Adv}_3^{C2} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$$

To sum up, we have that

$$\mathsf{Adv}_2^{C2} \le q_{\mathsf{M}}(\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}})$$

Case 3: $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathbf{true}$. In this case, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle CHALLENGE-A query. We further split this case into two subcases: when $\mathcal{A}$ queries the challenge oracle at CHALLENGE-A for encrypting $i^\star$'s message at epoch $t^\star$ whether $\Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big)$ holds, see Case 3.1, or, $\Big(\mathsf{safe\text{-}st}_{\mathtt{A}}(t^\star)$ **and** $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big)$ holds, see Case 3.2.

Case 3.1: $\Big(\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\Big)$
Game C3.1.3 This game is identical to **Game 2** except the following modification:

1. Whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} t+1$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered, then the challenger replaces $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in the following eSend algorithm in Line 19 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$.

We analyze $\mathcal{A}$'s advantage in winning **Game C3.1.3** by hybrid games.

Game $\mathsf{hy}.0$: This game is identical to **Game 2**. Thus, we have that

$$\mathsf{Adv}_2^{C3.1} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \le j \le q_{\mathsf{ep}})$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. When entering epoch j from $j-1$, if the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered for $\mathtt{P} = \mathtt{A}$ if j odd and $\mathtt{P} = \mathtt{B}$ if j even, then in the following eSend algorithm, the challenger replaces $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in Line 19 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$.

It is obvious that **Game** $\mathsf{hy}.q_{\mathsf{ep}}$ is identical to **Game C3.1.3**. Thus, we have that

$$\mathsf{Adv}_3^{C3.1} = \mathsf{Adv}_{\mathsf{hy}.q_{\mathsf{ep}}}$$

Let E denote the event that $\mathcal{A}$ can distinguish any adjacent hybrid games **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$. Note that the modification in every hybrid game is independent of the behavior of the previous game. Thus, we have that

$$\mathsf{Adv}_2^{C3.1} - \mathsf{Adv}_3^{C3.1} \le q_{\mathsf{ep}} \Pr[E]$$

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ in **Game** $\mathsf{hy}.j$ indicates that **Game** $\mathsf{hy}.(j-1)$ is identical to **Game** $\mathsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ is not triggered. Note also that $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ is not triggered only when $\mathsf{safe\text{-}ch_P}(\mathsf{flag}, j-1, n_{\neg\mathsf{P}})$, which further implies that one of the following conditions must hold: (1) $\mathsf{safe\text{-}st_P}(j-1)$, or (2) $\mathsf{flag} = \mathsf{good}$ Then, we consider each of the two cases.

Case $\mathsf{safe\text{-}st_P}(j-1)$: First, $\mathsf{safe\text{-}st_P}(j-1)$ means $(j-1), (j-2) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$. Moreover, $(j-1) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$ indicates that (1) the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} (j-2)$ in **Game** $\mathsf{hy}.(j-2)$ is not triggered, and (2) the state corruption on $\mathtt{P}$ is not invoked during epoch $(j-1)$ and $(j-2)$. According to hybrid game **Game** $\mathsf{hy}.(j-2)$, the value $\mathsf{st_P}.nxs$ sampled uniformly at random during sending the first message in epoch $(j-2)$. In other words, $\mathsf{st_P}.nxs$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 19 in Figure 3. By the prf security of KDF_2, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$, then there must exist an attacker that distinguish the keyed KDF_2 and a random function. Thus, it holds that

$$\Pr[E] \le \epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}}$$

Case $\mathsf{flag} = \mathsf{good}$: This means, the first message in epoch $j-2$ is computed using fresh randomness. In particular, this means, $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 19 in Figure 3 uses fresh randomness r. It is easy to know that $\mathsf{st_P}.nxs$ after sending the first message in epoch $(j-2)$ is distinguishable from a random string, due to the swap-security of KDF_2. Thus, we have that

$$\Pr[E] \le \epsilon_{\mathsf{KDF}_2}^{\mathsf{swap}}$$

From above two cases, we know that

$$\Pr[E] \le \max\left(\epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}} + \epsilon_{\mathsf{KDF}_2}^{\mathsf{swap}}\right) \le \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

To sum up, we have that

$$\mathsf{Adv}_2^{C3.1} \le q_{\mathsf{ep}} \Pr[E] + \mathsf{Adv}_3^{C3.1} \le \mathsf{Adv}_3^{C3.1} + q_{\mathsf{ep}} \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

Game C3.1.5, C3.1.6, C3.1.7. Note that $\mathsf{safe\text{-}st_B}(t^\star)$ means that $t^\star, (t^\star - 1) \notin \mathcal{L}_{\mathsf{B}}^{\mathsf{cor}}$. This implies that both following conditions must hold:

1. $\mathsf{st_P}.nxs \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ are executed when $\mathtt{B}$ was entering $t^\star - 1$.
2. The corruption oracle CORRUPT-B is not queried during $t^\star$ and $(t^\star - 1)$.

Furthermore, the KEM key pair in $\mathsf{st_B}$ generated in epoch $t^\star - 1$ for $\mathtt{A}$ to encrypt messages in $t^\star$ is not leaked. Applying a similar game hopping to the KEM key pair in the state, as to the identity key pairs in **Game 1.3, 1.4, 1.5**, we can easily have that

$$\mathsf{Adv}_3^{C3.1} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$$

Combing the above statements, we have that

$$\mathsf{Adv}_2^{C3.1} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

Case 3.2: $\Big(\textbf{safe-st}_{\mathtt{A}}(t^\star)$ **and** $\textbf{safe-st}_{\mathtt{B}}(t^\star)\Big)$

Game C3.2.3 This game is identical to **Game 2** except the following modification:

1. Whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} t+1$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered, then the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the following eSend algorithm in Line 22 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.rk \xleftarrow{\$} \{0,1\}^\lambda$ and $\mathsf{st}.ck^{\mathsf{st}.t} \xleftarrow{\$} \{0,1\}^\lambda$, followed by storing $(t+1, \mathsf{st}_{\mathtt{P}}.rk, \mathsf{st}.ck^{t+1}, \mathsf{st}.\mathsf{prtr})$.
2. if there exist a locally stored tuple $(t', rk, ck, \mathsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the eRcv algorithm in Line 33 in Figure 3 by $\mathsf{st}.rk \leftarrow rk$, $\mathsf{st}.ck^{\mathsf{st}.t} \leftarrow ck$.

We analyze $\mathcal{A}$'s advantage in winning **Game C3.2.3** by hybrid games.

Game $\mathsf{hy}.0$: This game is identical to **Game 2**. Thus, we have that

$$\mathsf{Adv}_2^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \leq j \leq q_{\mathsf{ep}})$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. When $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch j (i.e. $\mathtt{P} = \mathtt{A}$ if j odd and $\mathtt{P} = \mathtt{B}$ if t even) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered, then the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^j) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the following eSend algorithm in Line 22 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.rk \xleftarrow{\$} \{0,1\}^\lambda$ and $\mathsf{st}.ck^j \xleftarrow{\$} \{0,1\}^\lambda$, followed by storing $(j, \mathsf{st}_{\mathtt{P}}.rk, \mathsf{st}.ck^j, \mathsf{st}.\mathsf{prtr})$.
2. if there exist a locally stored tuple $(t', rk, ck, \mathsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^j) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the eRcv algorithm in Line 33 in Figure 3 by $\mathsf{st}.rk \leftarrow rk$, $\mathsf{st}.ck^j \leftarrow ck$.

It is obvious that **Game** $\mathsf{hy}.q_{\mathsf{ep}}$ is identical to **Game C3.1.3**. Thus, we have that

$$\mathsf{Adv}_3^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.q_{\mathsf{ep}}}$$

Let E denote the event that $\mathcal{A}$ can distinguish any adjacent hybrid games **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$. Note that the modification in every hybrid game is independent of the behavior of the previous game. Thus, we have that

$$\mathsf{Adv}_2^{C3.2} - \mathsf{Adv}_3^{C3.2} \leq q_{\mathsf{ep}} \Pr[E]$$

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in **Game** $\mathsf{hy}.j$ indicates that **Game** $\mathsf{hy}.(j-1)$ is identical to **Game** $\mathsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered. Note also that $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered only when $\mathsf{safe\text{-}ch}_{\mathsf{P}}(\mathsf{flag}, j-1, n_{\neg\mathsf{P}})$, which further implies that one of the following conditions must hold:

1. $\Big(\mathsf{safe\text{-}st}_{\mathsf{P}}(j-1)$ **and** $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\Big)$
2. $\Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\Big)$
3. $\Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe}_{\neg\mathsf{P}}^{\mathsf{idK}}\Big)$
4. $\Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe}_{\neg\mathsf{P}}^{\mathsf{preK}}(\mathsf{ind})\Big)$

Then, we consider each of the four cases:

Case $\Big(\mathsf{safe\text{-}st}_{\mathsf{P}}(j-1)$ **and** $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\Big)$**:** Recall that $\mathsf{safe\text{-}st}_{\mathsf{P}}(j-1)$ and $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)$ means $(j-1), (j-2) \notin \mathcal{L}_{\mathsf{A}}^{\mathsf{cor}}, \mathcal{L}_{\mathsf{B}}^{\mathsf{cor}}$. This indicates that (1) the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} (j-1)$ in **Game** $\mathsf{hy}.(j-1)$ is not triggered, and (2) the state corruption on both party is not invoked during epoch $(j-1)$. (3) the first message that P receives in the epoch $(j-1)$ is not forged by the attacker. According to hybrid game **Game** $\mathsf{hy}.(j-1)$, the value $\mathsf{st}_{\mathsf{P}}.rk$ sampled uniformly at random during sending the first message in epoch $(j-1)$. In other words, $\mathsf{st}_{\mathsf{P}}.rk$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $(\mathsf{st}.rk, \mathsf{st}.ck^j) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ is executed in the eSend algorithm in Line 22 in Figure 3. By the prf security of KDF_3, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$, then there must exist an attacker that distinguish the keyed KDF_3 and a random function. Thus, it holds that

$$\Pr[E] \leq \epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}$$

Case $\Big(\mathsf{flag} = \mathsf{good}$ **and** $\mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\Big)$ **:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game C3.1.3**, **Game C1.3**, **Game C1.4**, and **Game C1.5**. First, similar to analysis in **Game C3.1.3**, we know that KEM public key stored in $\mathsf{st}[\neg\mathsf{P}]$ and will be used by P in epoch j is sampled uniformly at random except probability $q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$. Next, similar to **Game C1.3**, we know that the encapsulated key is indistinguishable from a random key except probability $\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$ due to the $\mathsf{IND\text{-}CCA}$ security of the underlying KEM. Then, similar to **Game C1.4**, we know that the update value $\mathsf{upd}^{\mathsf{ar}}$ is indistinguishable from a random string in $\{0,1\}^\lambda$ except probability $\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$ due to the $\mathsf{3prf}$ security of the KDF_1. Finally, similar to **Game C1.5**, the root key $\mathsf{st}.rk$

and the chain key $\mathsf{st}.ck^j$ are indistinguishable from random strings except probability $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}$ due to the swap-security of the function KDF_5.
Thus, we have that

$$\Pr[E] \leq q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}$$

Case $\left(\mathsf{flag} = \mathsf{good}\ \textbf{and}\ \mathsf{safe}^{\mathsf{idK}}_{\neg\mathsf{P}}\right)$**:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game C1.3**, **Game C1.4**, and **Game C1.5**. First, similar to **Game C1.3**, we know that the encapsulated key is indistinguishable from a random key except probability $\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}$ due to the $\mathsf{IND\text{-}CCA}$ security of the underlying KEM. Then, similar to **Game C1.4**, we know that the update value $\mathsf{upd}^{\mathsf{ar}}$ is indistinguishable from a random string in $\{0,1\}^\lambda$ except probability $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$ due to the $\mathsf{3prf}$ security of the KDF_1. Finally, similar to **Game C1.5**, the root key $\mathsf{st}.rk$ and the chain key $\mathsf{st}.ck^j$ are indistinguishable from random strings except probability $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}$ due to the swap-security of the function KDF_5.
Thus, we have that

$$\Pr[E] \leq \epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}$$

Case $\left(\mathsf{flag} = \mathsf{good}\ \textbf{and}\ \mathsf{safe\text{-}st}_{\neg\mathsf{P}}(j-1)\right)$ **:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game C2.3**, **Game C2.4**, **Game C2.5**, and **Game C2.6**. First, similar to analysis in **Game C2.3**, the challenger first guesses the medium-term pre-key that will be used for sending the first message in epoch j, which can be guessed correctly with probability at least $\frac{1}{q_{\mathsf{M}}}$. Next, similar to **Game C2.4**, we know that the encapsulated key is indistinguishable from a random key except probability $\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}$ due to the $\mathsf{IND\text{-}CCA}$ security of the underlying KEM. Then, similar to **Game C2.5**, we know that the update value $\mathsf{upd}^{\mathsf{ar}}$ is indistinguishable from a random string in $\{0,1\}^\lambda$ except probability $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$ due to the $\mathsf{3prf}$ security of the KDF_1. Finally, similar to **Game C2.6**, the root key $\mathsf{st}.rk$ and the chain key $\mathsf{st}.ck^j$ are indistinguishable from random strings except probability $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}$ due to the swap-security of the function KDF_5.
Thus, we have that

$$\Pr[E] \leq q_{\mathsf{M}}(\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5})$$

From above two cases, we know that

$$\begin{aligned}\Pr[E] \leq & \max\Big(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5},\\ & \qquad \epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}, q_{\mathsf{M}}(\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5})\Big)\\ \leq & \max\Big(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5},\\ & \qquad q_{\mathsf{M}}(\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5})\Big)\end{aligned}$$

This means, it holds that

$$\mathsf{Adv}_2^{C3.2} \leq \mathsf{Adv}_3^{C3.2} + q_{\mathsf{ep}} \max \Big(\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}, q_{\mathsf{ep}} \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + \epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}, q_{\mathsf{M}}(\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + \epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + \epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5}) \Big)$$

Game C3.2.4. This game is identical to **Game 3.2.3** except the following modification:

1. For running $\mathtt{A}$'s eSend at $t^\star$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 23 in Figure 3 is replaced by $\mathsf{st}.ck^{t^\star} \xleftarrow{\$} \{0,1\}^\lambda$, $urk \xleftarrow{\$} \{0,1\}^\lambda$. After that, the challenger stored $(\mathsf{st}.ck^{t^\star}, urk)$ into a local list.
2. For running $\mathtt{B}$'s eRcv at $t^\star$ the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 38 is replaced by the tuple $(\mathsf{st}.ck^{t^\star}, urk)$ in the local list for the corresponding message index.

The advantage gap of $\mathcal{A}$ in winning **Game C3.2.3** and **Game C3.2.4** can be computed by hybrid games. Recall that $\mathcal{A}$ can query oracles at most q times, the maximum of the message index is q.

Game $\mathsf{hy}.0$: This game is identical to **Game C3.2.3**. Thus, we have that

$$\mathsf{Adv}_3^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \leq j \leq q)$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. For running $\mathtt{A}$'s j-th eSend at $t^\star$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 23 in Figure 3 is replaced by $\mathsf{st}.ck^{t^\star} \xleftarrow{\$} \{0,1\}^\lambda$, $urk \xleftarrow{\$} \{0,1\}^\lambda$. After that, the challenger stored $(\mathsf{st}.ck^{t^\star}, urk)$ into a local list.
2. For running $\mathtt{B}$'s eRcv on a ciphertext corresponds to the position $(t^\star, j)$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 38 is replaced by the tuple $(\mathsf{st}.ck^{t^\star}, urk)$ in the local list for the corresponding message index j.

It is obvious that **Game** $\mathsf{hy}.q$ is identical to **Game C3.2.4**. So, we have that $\mathsf{Adv}_4^{C3.2} = \mathsf{Adv}_{\mathsf{hy}.q}$. The gap between every two adjacent hybrid games can be reduced to the prg security of KDF_4. Namely, if the attacker can distinguish **Game** $\mathsf{hy}.(j-1)$ from **Game** $\mathsf{hy}.j$, then there must exist an attacker can distinguish the real KDF_4 and a random number generator. Thus, we can easily have that

$$\mathsf{Adv}_3^{C3.2} \leq \mathsf{Adv}_4^{C3.2} + q\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4}$$

Game C3.2.5. This game is identical to **Game C3.2.4** except the following modifications:

1. The challenger additionally samples a random message key $\widetilde{mk} \in \{0,1\}^\lambda$ for the position $(t^\star, i^\star)$
2. $c' \leftarrow \mathsf{S.Enc}(mk, m)$ in Line 23 and 39 in Figure 3 is replaced by $c' \leftarrow \mathsf{S.Enc}(\widetilde{mk}, m)$

Note that the unidirectional ratchet key urk is sampled random in **Game C3.2.4**. Similar to the game **Game C1.5**, if $\mathcal{A}$ can distinguish **Game C3.2.4** and **Game C3.2.5**, then we can construct an attacker that breaks prf security of underlying KDF_5. Thus, we have that

$$\mathsf{Adv}_4^{C3.2} \leq \mathsf{Adv}_5^{C3.2} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}}$$

Game Final Analysis for Case C3.2:

Similar to the final analysis for **Game C1**, if the attacker $\mathcal{A}$ can distinguish the challenge bit in **Game C3.2.5**, then there exists an attacker that breaks $\mathsf{IND\text{-}1CCA}$ security of the underlying SKE. Thus, we can easily have that

$$\mathsf{Adv}_5^{C3.2} \leq \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$$

To sum up, we have that

$$\begin{aligned}\mathsf{Adv}_2^{C3.2} \leq & q_{\mathsf{ep}} \max \Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}, q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}, \\ & q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big) + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}\end{aligned}$$

Combining all statements above, the proof is concluded by

$$\begin{aligned}& \mathsf{Adv}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{PRIV}} \\ \leq & q \max(\mathsf{Adv}_2^{C1}, \mathsf{Adv}_2^{C2}, \mathsf{Adv}_2^{C3.1}, \mathsf{Adv}_2^{C3.2}) \\ \leq & q \max \Bigg(\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}, \\ & q_{\mathsf{M}}(\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}), \\ & \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}, \\ & q_{\mathsf{ep}} \max \Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}, q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}, \\ & \quad q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big) + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} \Bigg) \\ \leq & q \Bigg(q_{\mathsf{M}}\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}} + q_{\mathsf{ep}}(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}) + \\ & q_{\mathsf{M}}q_{\mathsf{ep}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}) + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} \Bigg)\end{aligned}$$

D.4 Proof of Theorem 4

Proof. The proof is given by a sequence of games. Let Adv_i denote the attacker $\mathcal{A}$'s advantage in winning Game i. At the beginning of the experiment, the attacker $\mathcal{A}$ outputs a target epoch $t^\star$, such that it only queries the injection oracles inputting ciphertexts corresponding to in this epoch. Without loss of generality, we assume $t^\star$ is even, i.e., $\mathtt{A}$ is the message receiver. The case for $t^\star$ is even can be given analogously. Note also that the attacker $\mathcal{A}$ can immediately win when it successfully triggers the winning predicate $\mathsf{win}^{\mathsf{auth}}$ turning form false to true. So, we only consider the case that $\mathcal{A}$ successfully forges a ciphertext only once.

Game 0. This game is identical to the $\mathsf{Exp}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, we have that

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$$

Game 1. This game is identical to **Game 0** except the following modifications:

1. If the attacker queries $\textsc{Inject-A}(\mathsf{ind}, c)$ with c corresponding epoch $t^\star$ and a message index $i^\star$ such that $t^\star \leq t_{\mathtt{A}} - 2$ and $(\mathtt{B}, t^\star, i^\star) \notin \mathsf{trans}$, the challenger immediately aborts the oracle and outputs $(t^\star, i, \perp)$.

Note that a record is not included in the transcript set for the previous epochs, only when

1. this record is delivered
2. no sender has produced any message in the previous epoch $t^\star$ with message index $i^\star$

The first case can be easily excluded, since a natural eSM scheme never accepts two messages at the same position. For the second case, note that $\mathtt{B}$ produces messages only with continuous message indices. $\mathtt{B}$ didn't produce the message with message index $i^\star$ means that $i^\star$ exceeds the maximal message length that $\mathtt{B}$ has produced in the epoch $t^\star$. Since in eSM $\mathtt{A}$ has received all maximal message length in all previous epochs (see Line 31 in Figure 3) and will aborts the eRcv execution if i exceeds the maximal message length in the corresponding epoch (see Line 27 in Figure 3). This game is identical to **Game 0** from $\mathcal{A}$'s view. Thus, we have that

$$\mathsf{Adv}_1 = \mathsf{Adv}_0$$

Note that the attacker can win only when it queries $\textsc{Inject-A}(\mathsf{ind}, c)$ such that all of the following conditions hold

1. c corresponds to epoch $t^\star$
2. $(\mathtt{B}, c) \notin \mathsf{trans}$
3. $\mathsf{ind} \leq n_{\mathtt{A}}$
4. $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{A}}) = \mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{A}})$ and $\mathsf{safe\text{-}inj}_{\mathtt{A}}(t_{\mathtt{B}}) = \mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}})$
5. $m' \neq \perp$
6. $(\mathtt{B}, t^\star, i^\star) \notin \mathsf{comp}$

where $(\mathsf{st}_{\mathtt{A}}, t^\star, i^\star, m') \leftarrow \mathsf{eRcv}(\mathsf{st}_{\mathtt{A}}, ik_{\mathtt{A}}, prepk^{\mathsf{ind}}_{\mathtt{A}}, c)$

In particular, $(\mathtt{B}, t^\star, i^\star) \notin \mathsf{comp}$ but $(\mathtt{B}, t^\star, i^\star) \in \mathsf{trans}$ means that

1. $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$ holds at the time of $\mathtt{B}$ sending message corresponding to the position $(t^\star, i^\star)$, and

2. if $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$, CORRUPT-A cannot be queried
3. If CORRUPT-A is queried at epoch $t^\star$, then CORRUPT-B cannot be queried.
4. CORRUPT-B can be queried only after the ciphertext corresponding to $(t^\star, i^\star)$ has been honestly generated.
5. After the leakage of identity keys or pre-keys, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$

So, at most one of CORRUPT-A and CORRUPT-B at epoch $t^\star$, but not both. We separate the analysis for $t^\star \geq t_{\mathtt{A}} - 1$, see Case 1, or $t^\star \leq t_{\mathtt{A}} - 2$, see Case 2.

Case 1. $t^\star \geq t_{\mathtt{A}} - 1$

In this case, the attacker queries INJECT-A(ind, c) for some pre-key index ind and ciphertext c under the condition that $\mathsf{safe\text{-}st}_{\mathtt{B}}(t_{\mathtt{B}}) = \text{true}$. This means, $t_{\mathtt{B}}, (t_{\mathtt{B}} - 1) \notin \mathcal{L}_{\mathtt{B}}^{\mathsf{cor}}$.

Game C1.2 This game is identical to **Game 1** except the following modification:

1. Until epoch $t^\star$, whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} t+1$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered, then the challenger replaces $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in the following eSend algorithm in Line 19 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.nxs \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$.

We analyze $\mathcal{A}$'s advantage in winning **Game C1.2** by hybrid games.

Game $\mathsf{hy}.0$: This game is identical to **Game 1**. Thus, we have that

$$\mathsf{Adv}_1^{C1.1} = \mathsf{Adv}_{\mathsf{hy}.0}$$

Game $\mathsf{hy}.j, (1 \leq j \leq q_{\mathsf{ep}})$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1. When entering epoch j from $j-1$, if the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered for $\mathtt{P} = \mathtt{A}$ if j odd and $\mathtt{P} = \mathtt{B}$ if j even, then in the following eSend algorithm, the challenger replaces $r \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $(\mathsf{st}_{\mathtt{P}}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}_{\mathtt{P}}.nxs, r)$ executed in Line 19 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.nxs \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $r^{\mathsf{KEM}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $r^{\mathsf{DS}} \overset{\$}{\leftarrow} \{0,1\}^\lambda$.

It is obvious that **Game** $\mathsf{hy}.q_{\mathsf{ep}}$ is identical to **Game C1.2**. Thus, we have that

$$\mathsf{Adv}_2^{C1} = \mathsf{Adv}_{\mathsf{hy}.q_{\mathsf{ep}}}$$

Let E denote the event that $\mathcal{A}$ can distinguish any adjacent hybrid games **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$. Note that the modification in every hybrid game is independent of the behavior of the previous game. Thus, we have that

$$\mathsf{Adv}_1^{C1} - \mathsf{Adv}_2^{C1} \leq q_{\mathsf{ep}} \Pr[E]$$

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ in **Game** $\mathsf{hy}.j$ indicates that

Game $\mathsf{hy}.(j-1)$ is identical to **Game** $\mathsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ is not triggered. Note also that $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} j$ is not triggered only when $\mathsf{safe\text{-}ch_P}(\mathsf{flag}, j-1, n_{\neg \mathsf{P}})$, which further implies that one of the following conditions must hold: (1) $\mathsf{safe\text{-}st_P}(j-1)$, or (2) $\mathsf{flag} = \mathsf{good}$ Then, we consider each of the two cases.

Case $\mathsf{safe\text{-}st_P}(j-1)$**:** First, $\mathsf{safe\text{-}st_P}(j-1)$ means $(j-1), (j-2) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$. Moreover, $(j-1) \notin \mathcal{L}_{\mathsf{P}}^{\mathsf{cor}}$ indicates that (1) the execution $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} (j-2)$ in **Game** $\mathsf{hy}.(j-2)$ is not triggered, and (2) the state corruption on P is not invoked during epoch $(j-1)$ and $(j-2)$. According to hybrid game **Game** $\mathsf{hy}.(j-2)$, the value $\mathsf{st_P}.nxs$ sampled uniformly at random during sending the first message in epoch $(j-2)$. In other words, $\mathsf{st_P}.nxs$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $r \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 19 in Figure 3. By the prf security of KDF_2, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\mathsf{hy}.(j-1)$ and **Game** $\mathsf{hy}.j$, then there must exist an attacker that distinguish the keyed KDF_2 and a random function. Thus, it holds that

$$\Pr[E] \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}}$$

Case $\mathsf{flag} = \mathsf{good}$**:** This means, the first message in epoch $j-2$ is computed using fresh randomness. In particular, this means, $r \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 19 in Figure 3 uses fresh randomness r. It is easy to know that $\mathsf{st_P}.nxs$ after sending the first message in epoch $(j-2)$ is distinguishable from a random string, due to the swap-security of KDF_2. Thus, we have that

$$\Pr[E] \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{swap}}$$

From above two cases, we know that

$$\Pr[E] \leq \max\left(\epsilon_{\mathsf{KDF}_2}^{\mathsf{prf}} + \epsilon_{\mathsf{KDF}_2}^{\mathsf{swap}}\right) \leq \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

To sum up, we have that

$$\mathsf{Adv}_1^{C1} \leq q_{\mathsf{ep}} \Pr[E] + \mathsf{Adv}_2^{C1} \leq \mathsf{Adv}_2^{C1} + q_{\mathsf{ep}} \epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

Final Analysis for Case C1. Note that $t_{\mathsf{A}} - 1 \leq t^\star$ and that $t^\star$ even. Then, there are following seven cases:

1. t_{A} is even: $t_{\mathsf{A}} = t_{\mathsf{B}} = t^\star$
2. t_{A} is odd: $t^\star = t_{\mathsf{A}} - 1$, $t_{\mathsf{B}} = t_{\mathsf{A}} - 1$
3. t_{A} is odd: $t^\star = t_{\mathsf{A}} - 1$, $t_{\mathsf{B}} = t_{\mathsf{A}}$
4. t_{A} is odd: $t^\star = t_{\mathsf{A}} - 1$, $t_{\mathsf{B}} = t_{\mathsf{A}} + 1$
5. t_{A} is odd: $t^\star = t_{\mathsf{A}} + 1$, $t_{\mathsf{B}} = t_{\mathsf{A}} - 1$
6. t_{A} is odd: $t^\star = t_{\mathsf{A}} + 1$, $t_{\mathsf{B}} = t_{\mathsf{A}}$
7. t_{A} is odd: $t^\star = t_{\mathsf{A}} + 1$, $t_{\mathsf{B}} = t_{\mathsf{A}} + 1$

In all of above seven cases, $t^\star$ and t_{B} are not two epochs apart. Moreover, by $\mathsf{safe\text{-}st_B}(t_{\mathsf{A}})$ amd $\mathsf{safe\text{-}st_B}(t_{\mathsf{B}})$, we know that the $\mathcal{A}$ has to forge a signature against a pair of uncorrupted and freshly generated key pair, due to **Game C1.2**. To make a successful injection query, $\mathcal{A}$ has to either keep the pre-transcript and forge a signature for the pre-transcript or forge a signature for a new pre-transcript, which violates the $\mathsf{SUF\text{-}CMA}$ security of the underlying DS scheme. Thus, we can have that

$$\mathsf{Adv}_2^{C1} \leq \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}$$

To sum up, we have that

$$\mathsf{Adv}_1^{C1} \leq \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

Case 2. $t^\star \leq t_{\mathtt{A}} - 2$

In this case, $\mathcal{A}$ aims to forge a ciphertext in a past epoch. By **Game 1**, we know that $(t^\star, i^\star) \in \mathsf{trans}$, where $i^\star$ denotes the message index corresponding to the forged ciphertext.

Game C2.2 This game is identical to **Game 1** except the following modification:

1. The challenger directly outputs $(t^\star, i, \perp)$ for answering any Inject-A(ind, c) if $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$, where $(t^\star, i)$ is the position of c.

Note that $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$ holds at the time of $\mathtt{B}$ sending message corresponding to the position $(t^\star, i^\star)$ for some $i^\star$. This means, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$ when $\mathtt{B}$ was switch from receiver to sender when entering epoch $t^\star$. Similar to the analysis in **Game C1.2**, we know that the signing keys are randomly sampled except probability at most $q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$. If $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{true}$ at the time of any Inject-A query, the signing key has not been corrupted. Similar to the final analysis of Game C1.2, if $\mathcal{A}$ can forge a ciphertext, then we can construct another attacker that invokes $\mathcal{A}$ to break the $\mathsf{SUF\text{-}CMA}$ security of DS. Thus, we have that

$$\mathsf{Adv}_1^{C2} \leq \mathsf{Adv}_2^{C2} + \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$$

In the games below, we assume that $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \text{false}$ when $\mathcal{A}$ queries Inject-A. Recall that Corrupt-B can be queried only after the ciphertext corresponding to $(t^\star, i^\star)$ has been honestly generated. This also means that the unidirectional ratchet key urk for encrypting and decrypting the ciphertext corresponding position $(t^\star, i^\star)$ has been removed from the state $\mathsf{st}_{\mathtt{B}}$. Moreover, if Corrupt-B is queried, then Corrupt-A cannot be queried.

Game C2.3 This game is identical to **Game C2.2** except the following modification:

1. Until epoch $t^\star$. Whenever $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\mathtt{P} = \mathtt{A}$ if t even and $\mathtt{P} = \mathtt{B}$ if t odd) and the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \overset{+}{\leftarrow} t+1$ in Line 85 in the **ep-mgmt** helping function in Figure 3 is not triggered, then the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the following eSend algorithm in Line 22 in Figure 3 by $\mathsf{st}_{\mathtt{P}}.rk \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and $\mathsf{st}.ck^{\mathsf{st}.t} \overset{\$}{\leftarrow} \{0,1\}^\lambda$, followed by storing $(t+1, \mathsf{st}_{\mathtt{P}}.rk, \mathsf{st}.ck^{t+1}, \mathsf{st}.\mathsf{prtr})$.

2. if there exist a locally stored tuple $(t', rk, ck, \mathsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$ executed in the eRcv algorithm in Line 33 in Figure 3 by $\mathsf{st}.rk \leftarrow rk$, $\mathsf{st}.ck^{\mathsf{st}.t} \leftarrow ck$.

The analysis of this game is identical to **Game C3.2.3** in Section D.3. We can easily know that

$$\begin{aligned}\mathsf{Adv}_2^{C2} \leq & \mathsf{Adv}_3^{C2} + q_{\mathsf{ep}} \max \Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}, q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}, \\ & q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big)\end{aligned}$$

Game C2.4 This game is identical to **Game C2.3** except the following modification until CORRUPT-B is invoked:

1. For running $\mathtt{A}$'s eSend at $t^\star$, the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 23 in Figure 3 is replaced by $\mathsf{st}.ck^{t^\star} \xleftarrow{\$} \{0,1\}^\lambda$, $urk \xleftarrow{\$} \{0,1\}^\lambda$. After that, the challenger stored $(\mathsf{st}.ck^{t^\star}, urk)$ into a local list.
2. For running $\mathtt{B}$'s eRcv at $t^\star$ the execution $(\mathsf{st}.ck^{t^\star}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{t^\star})$ in Line 38 is replaced by the tuple $(\mathsf{st}.ck^{t^\star}, urk)$ in the local list for the corresponding message index.

The advantage gap of $\mathcal{A}$ in winning **Game C3.2.3** and **Game C3.2.4** can be computed by hybrid games and reduced to the prg security of KDF_4. Note that $\mathcal{A}$ can query at most q, we can easily have that

$$\mathsf{Adv}_3^{C2} \leq \mathsf{Adv}_4^{C2} + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$$

Game C2.5. In this game, the challenger guesses the message index $i^\star$ that $\mathcal{A}$ wants to attack. Note that $\mathcal{A}$ can query at most q times oracles. The challenger guesses correctly with probability at least $\frac{1}{q}$. Thus, we have that

$$\mathsf{Adv}_4^{C2} \leq q\mathsf{Adv}_5^{C2}$$

Game C2.6. This game is identical to **Game C2.5** except the following modifications:

1. The challenger additionally samples a random message key $\widetilde{mk} \in \{0,1\}^\lambda$ for the position $(t^\star, i^\star)$
2. If the pre-key index ind equals the one for producing ciphertext at position $(t^\star, i^\star)$ and the KEM ciphertext are same as produced before, the challenger replaces $c' \leftarrow \mathsf{S.Enc}(mk, m)$ in Line 23 and 39 in Figure 3 by $c' \leftarrow \mathsf{S.Enc}(\widetilde{mk}, m)$. Otherwise, the challenger samples another random key $\widetilde{mk}'$ $\in \{0,1\}^\lambda$ for decrypting ciphertext at location $(t^\star, i^\star)$.

Note that the unidirectional ratchet key urk is sampled random in **Game C2.4**. If $\mathcal{A}$ can distinguish **Game C2.5** and **Game C2.6**, then we can construct an attacker that breaks prf security of underlying KDF_5. Thus, we have that

$$\mathsf{Adv}_5^{C2} \leq \mathsf{Adv}_6^{C2} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}}$$

Game C2.7. This game is identical to **Game C2.6** except the following modifications:

1. If $\mathcal{A}$ queries Inject-A(ind, c) such that
   (a) c corresponds to the position $(t^\star, i^\star)$
   (b) ind does not equal the one for producing the ciphertext at position $(t^\star, i^\star)$ or the KEM ciphertexts included in c do not equal the ones in the original ciphertext at position $(t^\star, i^\star)$

   then the challenger simply returns $(t^\star, i^\star, \perp)$

The gap between **Game C2.6** and **Game C2.7** can be reduced to the $\mathsf{IND\text{-}1CCA}$ security of SKE. The reduction simulates **Game C2.6** honestly except for the Inject-A(ind, c) that is described above. In this case, the reduction forwards the symmetric key ciphertext to its decryption oracle for a reply m'. Then, the reduction returns $(t^\star, i^\star, m')$ to $\mathcal{A}$. If the challenge bit is 0, then the reduction simulates **Game C2.6** honestly, otherwise, it simulates **Game C2.7**. Thus, if $\mathcal{A}$ can distinguish **Game C2.6** and **Game C2.7**, then the reduction can easily distinguish the challenge bit. Thus, we have that

$$\mathsf{Adv}_6^{C2} \leq \mathsf{Adv}_7^{C2} + \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$$

Game C2.8. This game is identical to **Game C2.7** except the following modifications:

1. If $\mathcal{A}$ queries Inject-A(ind, c) such that
   (a) c corresponds to the position $(t^\star, i^\star)$
   (b) ind equals the one for producing the ciphertext at position $(t^\star, i^\star)$ and the KEM ciphertexts included in c equal the ones in the original ciphertext at position $(t^\star, i^\star)$

   then the challenger simply returns $(t^\star, i^\star, \perp)$

The gap between **Game C2.7** and **Game C2.8** can be reduced to the $\mathsf{IND\text{-}1CCA}$ security of SKE. The reduction simulates **Game C2.7** honestly except for the Transmit-B(m, r) and Inject-A(ind, c) that is described above.

For the Transmit-B(m, r) query, the reduction forwards m to its encryption oracle for a ciphertext c'. The rest of this oracle is honestly simulated.

For the Inject-A(ind, c) query, the reduction forwards symmetric key ciphertext in the c to its decryption oracle for a reply m'. Then, the reduction returns $(t^\star, i^\star, m')$ to $\mathcal{A}$.

If the challenge bit is 0, then the reduction simulates **Game C2.7** honestly, otherwise, it simulates **Game C2.8**. Thus, if $\mathcal{A}$ can distinguish **Game C2.7** and **Game C2.8**, then the reduction can easily distinguish the challenge bit. Thus, we have that

$$\mathsf{Adv}_7^{C2} \leq \mathsf{Adv}_8^{C2} + \epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$$

Final Analysis for Case C2: Note that no matter what kind of Inject-A(ind, c) query $\mathcal{A}$ asks, where c corresponds to the position $(t^\star, i^\star)$, the challenger always returns $(t^\star, i^\star, \perp)$ immediately, according to **Game C2.7** and **Game C2.8**. Thus, $\mathcal{A}$ can never win and we have that

$$\mathsf{Adv}_8^{C2} = 0$$

To sum up, we have that

$$\begin{aligned}\mathsf{Adv}_1^{C2} \leq & \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + q(\epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}) \\ & + q_{\mathsf{ep}} \max\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}, q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + \epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}}, \\ & \quad q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big) \\ \leq & \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}) \\ & + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big)\end{aligned}$$

The following equation concludes the proof.

$$\begin{aligned} & \mathsf{Adv}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{AUTH}} \\ \leq & \max(\mathsf{Adv}_1^{C1}, \mathsf{Adv}_1^{C2}) \\ \leq & \Bigg(\epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q_{\mathsf{ep}}\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}, \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}) \\ & + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big)\Bigg) \\ \leq & \epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}} + q(\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{prf}} + 2\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}) \\ & + q_{\mathsf{ep}}\Big(\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}} + (q_{\mathsf{ep}}+1)\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}} + q_{\mathsf{M}}(\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}} + \epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}} + \epsilon_{\mathsf{KDF}_5}^{\mathsf{swap}})\Big)\end{aligned}$$

D.5 Proof of Theorem 5

Proof. The proof is given by reduction. Namely, if there exists an attacker $\mathcal{A}$ that breaks the offline deniability for the composition of a DAKE scheme Σ and our eSM construction Π in Section 5.1, then we can always construct an attacker $\mathcal{B}$ that breaks the offline deniability of Σ in terms of [6, Definition 11].

We first define the function $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}$ and the function $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}$ for our eSM construction Π.

- $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}_{\mathsf{aid}}^{prek}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$: this algorithm inputs a key $K \in iss$, identity public keys $ipk_{\mathtt{A}}$ and $ipk_{\mathtt{B}}$, a list of private pre-keys $\mathcal{L}_{\mathsf{rid}}^{prek}$, the sender identity sid, the receiver identity rid, the accuser identity aid, and the defendant identity did, followed by executing the following steps:
  1. $\mathsf{st}_{\mathtt{A}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}A}(K)$
  2. $\mathsf{st}_{\mathtt{B}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}B}(K)$
  3. $\mathsf{st}_{\mathsf{Fake}} \leftarrow \Big((\mathsf{st}_{\mathtt{A}}, \mathsf{rid}), (\mathsf{st}_{\mathtt{B}}, \mathsf{sid})\Big)$
  4. **return** $\mathsf{st}_{\mathsf{Fake}}$
- $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}(\mathsf{st}_{\mathsf{Fake}}, ipk, prepk, m, \mathsf{sid}, \mathsf{rid}, \mathsf{ind})$: this algorithm inputs a fake state $\mathsf{st}_{\mathsf{Fake}}$, an public identity key ipk, a public pre-key $prepk$, a message m, a sender identity sid, a receiver identity rid, and a pre-key index ind, followed by executing the following steps:

```
1. Parse $\Big((\mathsf{st_A}, \mathsf{id_A}), (\mathsf{st_B}, \mathsf{id_B})\Big) \leftarrow \mathsf{st_{Fake}}$
2. **if** $\mathsf{id_A} = \mathsf{sid}$, **then**
   (a) $(\mathsf{st_A}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st_A}, ipk, prepk, m)$
   (b) copy all symmetric values in session state $\mathsf{st_A}$ to session state $\mathsf{st_B}$
   (c) If $\mathsf{st_A}.t$ is incremented in the above $\Pi.\mathsf{eSend}$ invocation, then extract
       the new verification key $\mathsf{vk}$ and new encryption key $\mathsf{ek}$ from $c$, followed
       by set $\mathsf{vk}$ and $\mathsf{ek}$ into $\mathsf{st_B}$
   (d) $\mathsf{st_{Fake}} \leftarrow ((\mathsf{st_A}, \mathsf{id_A}), ik_{\mathsf{rid}}, \mathcal{L}^{prek}_{\mathsf{rid}}, (\mathsf{st_B}, \mathsf{id_B}))$
3. **else**
   (a) $(\mathsf{st_B}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st_B}, ipk, prepk, m)$
   (b) copy all symmetric values in session state $\mathsf{st_B}$ to session state $\mathsf{st_A}$
   (c) If $\mathsf{st_B}.t$ is incremented in the above $\Pi.\mathsf{eSend}$ invocation, then extract
       the new verification key $\mathsf{vk}$ and new encryption key $\mathsf{ek}$ from $c$, followed
       by set $\mathsf{vk}$ and $\mathsf{ek}$ into $\mathsf{st_A}$
   (d) $\mathsf{st_{Fake}} \leftarrow ((\mathsf{st_A}, \mathsf{id_A}), (\mathsf{st_B}, \mathsf{id_B}))$
```

At the beginning of the experiment, the attacker $\mathcal{B}$ inputs a list $\mathcal{L}_{\mathsf{all}}$ that includes all public-private key pairs of Σ from its challenger. Next, $\mathcal{B}$ honestly samples the random identity key and pre-key pairs of Π and sets them into the respective lists as in the $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,\mathsf{n_P},q_{\mathsf{M}},\mathsf{n_S}}$. In particular, all public-private key pairs are added into the list $\mathcal{L}_{\mathsf{all}}$. $\mathcal{B}$ also initializes a empty dictionary $\mathcal{D}_{\mathsf{session}}$ and a counter n to 0. Then, $\mathcal{B}$ sends the list $\mathcal{L}_{\mathsf{all}}$ to $\mathcal{A}$.

When $\mathcal{A}$ queries $\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$, $\mathcal{B}$ first checks whether $\{\mathsf{sid}, \mathsf{rid}\} = \{\mathsf{aid}, \mathsf{did}\}$ and $\mathsf{sid} \neq \mathsf{rid}$ holds. It either condition does not hold, $\mathcal{B}$ simply aborts the oracle. Next, $\mathcal{B}$ increments the counter n, followed by adding $\{\mathsf{sid}, \mathsf{rid}\}$ into the dictionary $\mathcal{D}_{\mathsf{session}}[n]$. Then, $\mathcal{B}$ checks whether $\mathsf{aid} = \mathsf{sid}$. If the conditions holds, then $\mathcal{B}$ simply honestly runs Σ on the corresponding input and finally derives a key $K \in iss$ and a transcript T. Otherwise, $\mathcal{B}$ queries its challenge oracle with the input $(\mathsf{sid}, \mathsf{rid}, \mathsf{ind})$ for the key K and the transcript T. After that, $\mathcal{B}$ runs the above defined function $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}^{prek}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ for a fake state $\mathsf{st}^n_{\mathsf{Fake}}$. Finally, $\mathcal{B}$ returns the transcript to $\mathcal{A}$.

When $\mathcal{A}$ queries $\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, \mathsf{sessID}, \mathsf{ind}, m)$, $\mathcal{B}$ simply simulates $\mathsf{Session\text{-}Execute}$ as if the bit $\mathsf{b} = 1$.

At the end of the experiment, when $\mathcal{A}$ outputs a bit b', $\mathcal{B}$ then forwards it to its challenger.

Note that our $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}$ algorithm perfectly simulates the process of running $\Pi.\mathsf{eInit\text{-}A}$ and $\Pi.\mathsf{eInit\text{-}B}$. Moreover, we consider two cases for the queries to the $\mathsf{Session\text{-}Execute}$ oracle:

1. If the sender identity sid in the $\mathsf{Session\text{-}Execute}$ oracle query is $\mathsf{id_A}$. Note that when a party receives a message from the partner in our eSM construction Π, it only passively updates the symmetric state, and optionally update the verification key and encryption key from the partner. In this case, our $\mathsf{Fake}^{\mathsf{eSend}}_{\Pi}$ algorithm perfectly simulates the case that $\mathsf{id_A}$ sends messages to $\mathsf{id_B}$.

2. If the sender identity sid in the $\mathsf{Session\text{-}Execute}$ oracle query is $\mathsf{id_B}$. In this case, similar to the analysis above, our $\mathsf{Fake}_{\Pi}^{\mathsf{eSend}}$ algorithm also perfectly simulates the case that $\mathsf{id_B}$ sends messages to $\mathsf{id_A}$.

To sum up, in both cases $\mathcal{B}$ perfectly simulates $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,\mathsf{n_P},q_{\mathsf{M}},\mathsf{n_S}}$ to $\mathcal{A}$. Thus, $\mathcal{B}$ wins if and only if $\mathcal{A}$ wins. Obviously, the number of sessions at least as many as the number of challenge oracles that $\mathcal{B}$ queries. And $\mathcal{A}$ and $\mathcal{B}$ runs in the approximately same time, which concludes the proof.

Table of Contents