Secure Messaging with Strong Compromise Resilience, Temporal Privacy, and Immediate Decryption

Abstract— **Recent years have seen many advances in designing secure messaging protocols, aiming at provably strong security properties in theory or high efficiency for real-world practical deployment. However, important trade-off areas of the design space inbetween these elements have not yet been explored.**

In this work we design the first provably secure protocol that at the same time achieves (i) strong resilience against fine-grained compromise, (ii) temporal privacy, and (iii) immediate decryption with constant-size overhead, notably, in the post-quantum (PQ) setting. Besides these main design goals, we introduce a novel definition of offline deniability suitable for our setting, and prove that our protocol meets it, notably when combined with a PQ offline deniable initial key exchange.

1. Introduction

Driven by the global uptake of the Signal protocol, which has been widely deployed in many messaging applications worldwide by virtue of its high efficiency and strong security guarantees, there have been many advances in the theory and design of messaging protocols with desirable efficiency and security properties during the last decade. We highlight three of these properties.

(i) Immediate Decryption with Constant-Size Overhead: This property, which is essential for practical messaging apps and was formally studied by Alwen et al. [1], requires that the recipients can decrypt every message at the time of arrival, irrespective of the arrival of prior messages. Conventional messaging solutions reuse a static encryption/decryption key pair during every two-party conversation (aka. session). However, the leakage of the private decryption keys indicates the loss of privacy of all messages in the past and/or future. Two basic security properties are formalized for modern messaging protocols: forward secrecy (FS) and post-compromise security (PCS). While FS requires the privacy of past messages prior to the state expose, PCS enables the parties to recover from state exposure. Common modern messaging solutions obtain strong security guarantees by making their encryption keys dependent in some way on all previously sent messages. However, in realistic messaging settings, messages can arrive out-of-order or may be lost forever. If message n arrives before message $n-1$, it cannot be decrypted until message $n-1$ arrives; and if it never arrives, communications become stuck. In theory, this can be naively solved by appending all previous ciphertexts to the next message sent. In practice, this naive solution is unusable, as practical applications require constant-size overhead for messages. The Signal protocol is a pioneering example in the domain of messaging with relatively strong security and immediate decryption with constant overhead.

(ii) Temporal Privacy: State compromise does not cause loss of privacy of messages sent prior to a time interval and can be healed after every time interval. Pijnenburg and Pöttering [2] first observe that the immediate decryption restricts FS by definition: an attacker that intercepts a message and corrupts the receiver in the future can always compromise this message. To solve this, [2] proposes a time-based BOOM protocol that expires old keys and updates new keys after a specific time interval. Intuitively, this solves the restricted FS problem as attackers cannot corrupt the expired keys that have been erased from the state. However, every party in BOOM obtains the partner's latest public key only when receiving the partner's latest message. If two parties do not frequently exchange messages, the restricted FS problem remains. A trivial fix is to force every party to frequently send "empty messages" for key updates. However, due to the key-updatable framework underlying BOOM, this solution potentially yields linearly growing bandwidth.

The original Signal protocol satisfies a similar temporal privacy property but only for new conversations. Conceptually, the Signal protocol defines the initial *Extended Triple-Diffie-Hellman* (X3DH) asynchronous key exchange [3] and the *Double Ratchet* (DR) [4] for the subsequent message exchanges. Note that the X3DH key establishment uses the combination of public/private keys with different lifetimes, i.e., long-term, medium-term, and one-time. Even if all previous keys are compromised, the privacy of new conversations can still be recovered if the honest recipients upload their new medium-term keys. Conversely, the privacy of all past conversations under a certain medium-term key holds if that key is not leaked, even if other keys are leaked.

(iii) Resilience against Fine-Grained State Compromise: The compromise of senders' and recipients' state does not cause loss of privacy and authenticity, respectively. Modern secure messaging protocols like Signal [5] have been fundamentally designed to be resilient against a weak form of state compromise: The state is healed from compromise after a back-and-forth interaction, i.e., PCS. However, Alwen et al. [1] notice that such state compromise resilience of Signal is very coarse rather than "fine-grained": corruption of the state of either party in a conversation will cause the loss of both privacy and authenticity, since the privacy and authenticity of messages depend on a symmetric secret that is present in both parties' states. It is however possible to achieve the stronger notion of resilience against fine-grained compromise by breaking this symmetry: in the literature, a number of "optimal-secure" protocols [2], [6]–[9] provably achieve such resilience against fine-grained compromise.

Challenges: Perhaps surprisingly, while each of the above properties have been studied in isolation, there currently exists no provably secure protocol that simultaneously offers the above three desirable properties.

Alwen et al. [1] generalize DR of Signal to a new SM protocol, based on which another TR protocol [10] is proposed with slightly stronger security. However, the original Signal, SM, and TR all satisfy immediate decryption with constant-size overhead but lack the resilience against fine-grained state compromise. To the best of our knowledge, the BOOM protocol [2] is the only known protocol that provides the temporal privacy. Moreover, similar to other "optimal-secure" protocols [6]–[9] in the literature, the BOOM protocol also provides a flavor of very strong security guarantee (we call it "ID-optimal") that includes the resilience against fine-grained state compromise. However, all these optimally secure protocols lack immediate decryption with constant-size overhead. We summarize the situation for related provably secure protocols in Figure 1.

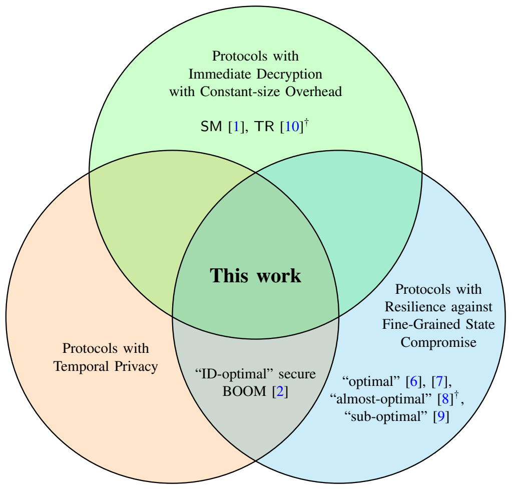

Figure 1: Comparison between this work and other existing protocols with provable security properties w.r.t. (i) immediate decryption with constant-size overhead, (ii) temporal privacy, and (iii) resilience against fine-grained state compromise. All constructions in the diagram (including this work) are PQ-compatible except for the ones marked with †.

Contributions: Our main contribution is the first provably secure messaging protocol with immediate decryption and constant-size overhead, temporal privacy, and resilience against fine-grained state compromise. To this end, we introduce a related new strong security notion called Extended-Secure-Messaging (eSM). We show that the eSM notion covers above strong properties and prove that our protocol meets it, in particular, in the PQ setting.

Furthermore, to show that our protocol is a suitable PQ-secure candidate for the DR in Signal, which is provably offline deniable, we extend the offline deniability definition for SPQR [11] (currently the only provably secure PQ-asynchronous key establishment) to the multi-stage setting. We prove that the combination of our eSM-secure protocol and SPQR is offline deniable, making it the first full messaging protocol that is provably offline deniable in the PQ setting.

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

We recall related cryptographic primitives in Appendix F. 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 with Constant-Size Overhead

The Signal protocol provably offers strong security guarantees, such as *forward secrecy* and *post-compromise security* [5], [12], and *offline deniability* [13]. Moreover, Signal has several features that are critical 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. Notably, the Signal protocol provides the above properties with constant-size overhead.

The core Signal protocol consists of two components: the *Extended Triple-Diffie-Hellman* (X3DH) initial key exchange and the *Double Ratchet* (DR) for subsequent message transmissions. Alwen et al. [1] introduce the notion of *Secure Messaging* (SM), which is a syntax and associated security notion that generalizes the security of Signal's DR. Alwen et al. also provide a concrete construction and prove that it is SM-secure. This construction is not explicitly named in [1]: in this work, we will refer to it as ACD19.

To the best of our knowledge, in addition to ACD19, the only known provably secure protocol that provides immediate decryption with constant-size overhead is the *Triple Ratchet* (TR) protocol [10]. However, the TR protocol is neither PQ-secure nor resilient against the fine-grained state compromise. We review the ACD19 and TR in details in Appendix A. For the interested readers, we also compare ACD19 and TR with our protocol in Appendix D.

2.2. Secure Messaging Protocols and Strong Security Guarantees

Alwen et al. [1] observe that the ACD19 protocol lacks resilience against fine-grained state compromise, because both encryption and decryption of a message in ACD19 uses the shared state of both parties in a conversation. The corruption of the shared state of either party immediately compromises the subsequent messages, no matter whether the corrupted party is the sender or receiver. To reduce the impact of state exposure, the authors also describe a second security notion for secure messaging, called PKSM, and a corresponding construction, which we call ACD19-PK. At a very high level, ACD19-PK extends ACD19 by encrypt-then-signing the output of the original SM protocol using a public key encryption (PKE) and a digital signature (DS). Intuitively, ACD19-PK reduces the impact of state compromise, since the attacker can neither recover the output of SM protocol (and further the real message) from the PKE ciphertext without knowing the recipient's decryption key, nor forge a valid ciphertext without knowing the sender's signing key. However, the main focus of [1] are SM and ACD19: for ACD19-PK, neither a formal security model nor a concrete proof is given; thus, its security is essentially conjectured.

In a parallel line of research, several messaging protocols have been proposed to meet various strong or even "optimal" security [2], [6]–[9], [14], [15]. They follow different ratcheting frameworks aiming at various flavors of security, notably, all of which capture resilience against fine-grained compromise. Unfortunately, none of them provides immediate decryption with constant-size overhead, due to their key-update or state-update structures.

In particular, [2] observes that a protocol satisfying immediate decryption can only achieve a weak form of forward secrecy: an attacker that intercepts a message and corrupts the receiver in the future can always compromise this message. To solve this, [2] proposes a novel strong security model, which we call "ID-optimal", and a time-based BOOM protocol that periodically expires old keys and updates new keys. By this, neither the receiver nor an attacker who corrupts the receiver's state can decrypt a message that was encrypted under an expired key. The efficiency and security can be balanced by picking a reasonable time interval for key update and expiration. However, we find that the BOOM protocol has two constraints: On the one hand, every party in BOOM obtains the partner's latest public key only at the time of receiving the partner's latest message. If the message exchange between two parties are not frequent, then the restricted forward secrecy problem remains. On the other hand, the BOOM protocol also makes use of a complicated key-update mechanism and therefore provides immediate decryption with linearly growing bandwidth.

We review protocols that meet various "optimal" security in Appendix B.

2.3. 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. The formal definition of offline deniability originates from [16] and [13] in the simulation-based models respectively for the authenticated key exchange (AKE) and full messaging protocols. These works also prove that several well-known classical AKE constructions, such as MQV, HMQV, 3DH, and X3DH, and the full Signal protocol are offline deniable.

Constructing PQ secure asynchronous key establishments is surprisingly complicated. There are a number of key establishment protocols [17]–[20] that are potential candidates for PQ security. However, all of their security proofs rely on either the random oracle model or novel tailored assumptions, which are still not well-studied in the PQ setting. Hashimoto et al. [21] propose the first PQ secure key establishment but unfortunately have to assume that every party can pre-upload inexhaustible one-time keys for full asynchronicity. A subsequent work by Brendel et al. [11] proposed a new PQ asynchronous deniable authenticated key exchange (DAKE) protocol, called SPQR, and a new game-based offline deniability notion. Brendel et al. prove that SPQR is offline deniable in the game-based paradigm against quantum (semi-honest) attackers.

To the best of our knowledge, SPQR is the only known PQ secure key establishment with full asynchronicity. Although it is straightforward that the combination of SPQR and ACD19 can form a PQ-secure full messaging protocol with promising privacy and authenticity, it is still an open question which flavors of offline deniability can be provably obtained for the combined protocols in the PQ setting.

3. Extended Secure Messaging

In this section, we first define our new *extended secure messaging* (eSM) scheme in Section 3.1, followed by the expected security properties in Section 3.2. Then, we define an associated strong security model (eSM) in Section 3.3.

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. Let $(\cdot)$ and $\{\cdot\}$ respectively denote an ordered tuple and an unordered set. For any positive integer n, let $[n]$ denote the set of integers from 1 to n, i.e., $[n] = \{1, ..., n\}$. We write $y \leftarrow A(x)$ for running a deterministic algorithm A with input x and assigning the output to y. 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 $n{+}{+}$ be a shorthand for $n \leftarrow n + 1$. We use _ to denote a value that is irrelevant. We use $\mathcal{D}$ to denote a dictionary that stores values for each index and $\mathcal{D}[\cdot] \leftarrow \perp$ for the dictionary initialization. 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.

3.1. 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*

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

Our eSM re-uses two important concepts *epoch* and *message index* that originate in [1].

Epoch. The epoch t is used to describe how many back-and-forth interactions in a two-party communication channel (aka. session) have been processed. Let $t_\mathtt{A}$ and $t_\mathtt{B}$ respectively denote the epoch counters 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 counter $t_\mathtt{P}$ is incremented by 1. In 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 sender and $\mathtt{A}$ acts as the receiver, and odd epochs in reverse. In each epoch, the sender can send arbitrarily many 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 index of a message in each epoch. Notably, the epoch number t and message index i output by eRcv indicate the position of the decrypted message m during the communication. The receiver is expected to recover the position of each decrypted message even if it is delivered out of order.

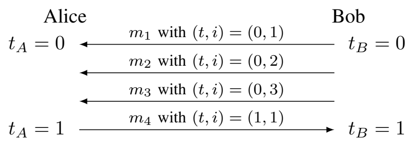
Alice
Bob
$t_A = 0$
$t_B = 0$
m_1 with $(t, i) = (0, 1)$
m_2 with $(t, i) = (0, 2)$
m_3 with $(t, i) = (0, 3)$
m_4 with $(t, i) = (1, 1)$
$t_A = 1$
$t_B = 1$

Figure 2: An example session between Alice and Bob. The session starts with $t_\mathtt{A} = t_\mathtt{B} = 0$, i.e., Bob is the sender. When Bob continuously sends messages, the message index grows from 1 for m_1 to 3 for m_3. When Alice switches the role from receiver to sender, the epoch increases to $t_\mathtt{A} = t_\mathtt{B} = 1$.

3.2. Strong Security Properties

The eSM schemes aim at following strong security properties. First, we expect our eSM to meet well-studied basic properties below:

1) **Correctness:** The messages exchanged between two parties are recovered in the correct order, if no attacker manipulates the underlying transmissions.
2) **Immediate decryption** (ID) **and message-loss resilience** (MLR)**:** Messages must be decrypted to the correct position as soon as they arrive; the loss of some messages does not prevent subsequent interaction.
3) **Forward secrecy** (FS): All messages that have been sent and received prior to a session state compromise of either party (or both) remain secure to an attacker.
4) **Post-compromise security** (PCS)**:** The parties can recover from session state compromise (assuming the access to fresh randomness) when the attacker is passive.

Second, our eSM targets the following advanced security against fine-grained compromise.

5) ***Strong*** **authenticity:** The attacker cannot modify the messages in transmission or inject new ones, unless the sender's session state is compromised.
6) ***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 corresponding private pre-key all are compromised.
7) **Randomness leakage/failures**: While both parties' session states are uncompromised, all above security properties (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.

Finally, our eSM also pursues two new security properties:

8) ***State compromise/failures*****:** While the sender's randomness quality is good and the receiver's private identity key or pre-key is not leaked, the privacy of the messages holds even if both parties' session states are corrupted.
9) ***Periodic privacy recovery*** (PPR)**:** If the attacker is passive (i.e., does not inject corrupted messages), the message privacy recovers from the compromise of both parties' all private information after a time period (assuming each has access to fresh randomness).

We stress that the first new property *state compromise/failures* has a particular impact for the secure messaging after an *insecure* key establishment. For instance, consider that the party $\mathtt{B}$ initializes a conversation with $\mathtt{A}$ using X3DH in Signal. The leakage of the sender $\mathtt{B}$'s private identity key and ephemeral randomness in X3DH implies the compromise of the initial shared secret and further both parties' session states in DR. If $\mathtt{B}$ continuously sends messages to $\mathtt{A}$ without receiving a reply in Signal, all messages in the sequence are leaked, since the attacker can use $\mathtt{A}$'s session state to decrypt the ciphertexts. An eSM protocol with the "state compromise/failures" property is able to prevent such attack.

Moreover, the second new property PPR complements the strong privacy. Assuming the secure randomness, the strong privacy ensures the secrecy of *past* messages if the corresponding private pre-keys are not leaked, while PPR ensures the secrecy of *future* messages if new pre-key pairs are randomly sampled and honestly delivered to the partner.

3.3. 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 3.

Notation. Our model considers the communication between two distinct parties $\mathtt{A}$ and $\mathtt{B}$. 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 in 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 tuple x includes y, i.e., $X(y) = \{x \in X \mid y \in x\}$. We say $y \in X$ if 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 an *authenticated* channel between each party and the server for key-update and -fetch and therefore no forgery of the public identity keys and pre-keys. This is the common treatment in the security analyses in this domain, e.g. [5], the server is considered to be a bulletin board, where each party can upload their own and fetch other parties' honest public keys. For practical deployments, we require that the key-upload and key-fetch processes between each party and sever use fixed bandwidth and are only executed periodically. We omit the discussion on the frequency of the pre-keys' upload and retrieve[1].

We assume that all session-specific data is stored at the same security level in the state, but the non-session-specific data that can be potentially shared among multiple sessions (i.e., identity keys and pre-keys) might be stored differently. Thus, corruption of session-specific state does not imply leakage of the private identity key and pre-key and vice versa. In fact, as we will show later, 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 a Hardware Security Module (HSM).

Moreover, we also require the eSM scheme Π to be *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* $m = \bot$,
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 must be rejected,*
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:

- $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}, \mathsf{safe}^{\mathsf{idK}}_{\mathtt{B}} \in \{\mathsf{true}, \mathsf{false}\}$: the boolean values indicating whether the private identity keys are revealed.

1. As an example, we can consider a scenario where every party is only allowed to upload and fetch public keys at 12am every day.

- $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}$: the lists that record the indices of the pre-keys that are revealed.
- $\mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{B}}$: the lists that record the indices of the epochs where the session states are corrupted.
- $n_{\mathtt{A}}, n_{\mathtt{B}}$: the pre-key counters.
- $t_{\mathtt{A}}, t_{\mathtt{B}}$: 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 undelivered yet, and their related information. See the helper function **record** for more details.
- allTrans: a set that records all honest encrypted ciphertexts (including both the delivered and undelivered ones), and their related information.
- chall: a set that records all challenge ciphertexts, which are honestly encrypted but undelivered yet, and their related information.
- 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.
- $\mathsf{win}^{\mathsf{corr}}, \mathsf{win}^{\mathsf{auth}}, \mathsf{win}^{\mathsf{priv}} \in \{\mathsf{true}, \mathsf{false}\}$: the winning predicate that indicates whether the attacker wins.
- $\mathsf{b} \in \{0, 1\}$: the challenge bit.

Moreover, the experiment $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ also includes four predicates as shown in Figure 3.

- $\mathsf{safe}^{\mathsf{preK}}_{\mathtt{P}}(\mathsf{ind})$: indicating whether ind-th pre-key of party $\mathtt{P}$ is leaked. We define it true if ind is not included in $\mathcal{L}^{\mathsf{rev}}_{\mathtt{P}}$.
- $\mathsf{safe\text{-}st}_{\mathtt{P}}(t)$: indicating whether the state of party $\mathtt{P}$ at epoch t is expected to be safe. This predicate simplifies the definition of $\mathsf{safe\text{-}ch}_{\mathtt{P}}$ and $\mathsf{safe\text{-}inj}_{\mathtt{P}}$ predicates below. We define it true if none of epochs from t to $(t - \triangle_{\mathsf{eSM}} + 1)$ is included in the list $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}}$.
- $\mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, t, \mathsf{ind})$: indicating whether the privacy of the message sent by $\mathtt{P}$ is expected to hold, under the randomness quality $\mathsf{flag} \in \{\mathsf{good}, \mathsf{bad}\}$, the sending epoch t, and the receiver $\neg\mathtt{P}$'s pre-key index ind. We define it to be true 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 ind-th 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. We define it to be true if the partner's state is safe at epoch t.

Helper Functions. To simplify the security experiment definition, we use five helper functions.

- **sam-if-nec**(r): If $r \neq \bot$, this function outputs (r, bad) indicating that the randomness is attacker-controlled. Otherwise, a new random string r is sampled from the space

$\mathcal{R}$[2] and is output together with a flag good.

- $\mathbf{record}(\mathtt{P}, \mathsf{type}, \mathsf{flag}, \mathsf{ind}, m, c)$: A record rec, which includes the party's identity $\mathtt{P}$, the partner's pre-key index ind, 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 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 type $=$ chall, the record rec is also added into the challenge sets chall and allChall.
- $\mathbf{ep\text{-}mgmt}(\mathtt{P}, \mathsf{flag}, \mathsf{ind})$: When the party $\mathtt{P}$ enters a new epoch as the sender upon the partner's ind-th pre-key, the new epoch number is added to the state corruption list $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}}$ 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 set to 0.
- $\mathbf{delete}(t, i)$: deletes all records that includes (t, i) from the sets trans, chall, and comp.
- $\mathbf{corruption\text{-}update}()$: checks all records in the allTrans 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.

Notably, the helper function **corruption-update** is invoked in the key-revealing and state-corruption oracles to capture the impact of the leakage of any secret on the secrecy of the (past) session states.

Experiment Execution and Oracles. At the beginning of the $\mathsf{Exp}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ security model, the safe predicates for identity keys, the reveal and corruption lists for pre-keys and states, and the pre-key counters are initialized. 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. 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 eInit-A and eInit-B of eSM. After initializing the epoch and message index counters, the sets, and the winning predicates $\mathsf{win}^{\mathsf{corr}}$ and $\mathsf{win}^{\mathsf{auth}}$, a challenge bit b is randomly sampled. The attacker is given access to all eighteen oracles and terminates the experiment by outputting a bit b' for evaluating the winning predicate $\mathsf{win}^{\mathsf{priv}}$. Finally, the experiment outputs all these three winning predicates. In Figure 3, we only depict nine oracles with suffix -A for party A. The oracles for party B are defined analogously.

Oracle Category 1: Identity and pre-keys. The first eight oracles are related to the generation and the leakage of identity keys and pre-keys.

- $\textsc{NewIdKey-A}(r)$, $\textsc{NewIdKey-B}(r)$: Both oracles can be queried at most once. 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 $\mathcal{L}_{\mathtt{A}}^{\mathsf{rev}}$ or $\mathcal{L}_{\mathtt{B}}^{\mathsf{rev}}$ if the input $r \neq \perp$. The public key is returned.
- $\textsc{RevIdKey-A}$, $\textsc{RevIdKey-B}$: These oracles simulate the reveal of the identity private key of a party $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$. The corresponding safe predicate is set to false. Then, the **corruption-update** helper function is invoked to update whether the current and past states are still secure or not. We require that this oracle invocation does not cause the change of safe challenge predicate for any record in the all-challenge set allChall. Otherwise, this oracle undoes all actions during this invocation and exits. This step prevents the attacker from distinguishing the challenge bit by trivially revealing enough information to decrypt the past challenge ciphertexts.
  Then, all records in the transcript set trans, whose safe injection predicate turns to false, are 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.
- $\textsc{RevPreKey-A}(n)$, $\textsc{RevPreKey-B}(n)$: These oracles simulate the reveal of the n-th private pre-key of a party $\mathtt{P}$. The input n must indicate a valid prekey counter, i.e., $n \leq n_{\mathtt{P}}$, and is added into the reveal list $\mathcal{L}_{\mathtt{P}}^{\mathsf{rev}}$. The rest of these oracles are same as above: (1) runs **corruption-update**, (2) aborts the oracles if the safe challenge predicates of any record in the allChall set is violated, and (3) adds all records in the trans set, whose safe injection predicate is violated, into the set comp.
  Finally, the corresponding private pre-key is returned.

Oracle Category 2: State Corruption. The following two oracles allow attackers to corrupt session states.

- $\textsc{Corrupt-A}$, $\textsc{Corrupt-B}$: These oracles simulate the corruption of party $\mathtt{P}$'s session states. First, the current epoch counter is added to the state corruption list $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}}$, followed running **corruption-update** to update whether this corruption impacts the safety of other session states. Next, we require that either the set chall does not include the record produced by the partner $\neg\mathtt{P}$, or such a record exists but (1) the flag in the record is good and (2) $\mathtt{P}$'s identity key or $\mathtt{P}$'s pre-key corresponding to the pre-key index in the record is safe. If the requirement is not satisfied, this oracle undoes all actions in this invocation and exits. This requirement prevents the attacker from distinguishing the challenge bit by trivially revealing enough information to decrypt the past challenge ciphertexts.
  After that, we add all records rec $\in$ trans, which are produced by $\neg\mathtt{P}$ at an unsafe epoch t, into the compromise set comp. We also add all records rec $\in$ 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 by corrupting the sender's state and forging the corresponding undelivered messages.
  Finally, the session states are returned.

Oracle Category 3: Message Transmission. The final eight oracles simulate the honest message encryptions and the attacker's capability of manipulating the message transmission.

2. The randomness space $\mathcal{R}$ is not specific and depends on the concrete functions and algorithms. Here, we use $\mathcal{R}$ only for simplicity.

- $\textsc{Transmit-A}(\mathsf{ind}, m, r)$, $\textsc{Transmit-B}(\mathsf{ind}, m, r)$: These transmission oracles simulate the real sending execution. The input index ind must not exceed the partner's current pre-key counter. 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 upon the partner's identity key and ind-th pre-key. After recording the transcript, the ciphertext is returned.
- $\textsc{Challenge-A}(\mathsf{ind}, m_0, m_1, r)$, $\textsc{Challenge-B}(\mathsf{ind}, m_0, m_1, r)$: These challenge 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 execution of transmission oracles with input $(\mathsf{ind}, m_\mathsf{b}, r)$ for the challenge bit $\mathsf{b} \in \{0, 1\}$ sampled at the beginning of the experiment. The only difference is that the safety predicate $\mathsf{safe\text{-}ch_P}(\mathsf{flag}, t_\mathtt{P}, \mathsf{ind})$ 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.
- $\textsc{Deliver-A}(\mathsf{ind}, c)$, $\textsc{Deliver-B}(\mathsf{ind}, c)$: These delivery oracles simulate the receiving execution of a ciphertext generated by the honest party. This means, there must exist a record $(\mathtt{P}, \mathsf{ind}, t, i, m, c)$ 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 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 the ciphertext c is considered as a forgery after this delivery. Finally, the output epoch t', the message index i', and the decrypted message m' is are returned.
- $\textsc{Inject-A}(\mathsf{ind}, c)$, $\textsc{Inject-B}(\mathsf{ind}, c)$: These oracles simulate a party P's receiving execution of a ciphertext forged by the attacker. The input $\mathsf{ind} \leq n_\mathtt{P}$ specifies a pre-key for running eRcv and the input c must be not produced by the partner in the transcript set. We require that eRcv is invoked under the condition that the safety predicates $\mathsf{safe\text{-}inj_P}(t_\mathtt{A})$ and $\mathsf{safe\text{-}inj_P}(t_\mathtt{B})$ both are true. If the decrypted message is not $\perp$ and the ciphertext at the same position is not compromised, the attacker wins via the $\mathsf{win}^{\mathsf{auth}}$ predicate. The rest of this oracle is identical to the delivery oracles.

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 attackers against the* $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ *experiment in Figure 3 in time t is bounded by*

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

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

Conclusion. Finally, we explain how our eSM security captures all security properties listed in Section 3.2.

- **Correctness:** No correctness means the encrypted message cannot be recovered correctly and causes the winning event via Line 48.
- **Immediate decryption and message-loss resilience:** No immediate decryption or message-loss resilience means that some messages cannot be recovered to the correct position from the delivered ciphertext when the attacker invokes the transmission and delivery oracles in an arbitrary order, which causes the winning event via Line 48.
- **Forward secrecy**: Note that the attacker can freely access the corruption oracles if all challenge ciphertexts have been delivered. No FS means that the attacker can distinguish the challenge bit from the past encrypted messages and wins via Line 12.
- **Post-compromise security:** Note that the states are not leaked to a passive attacker after the owner sends a reply in a new epoch (i.e., epochs are not added into the state corruption list in Line 88), assuming fresh randomness and the partner's uncorrupted state, or identity key or pre-key, see Line 87.
  No PCS indicates that a state at an epoch not in the state corruption lists might still be corrupted, which causes the lose of other security properties.
- **Strong authenticity:** The attacker can inject a forged ciphertext (Line 53) that does not correspond to a compromised ciphertext position (Line 56) 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 23, 36, 72, 75, 84).
  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 57.
- **Strong privacy**: Note that the challenge ciphertexts must be produced without the violation the safety predicate safe-ch in Line 64, i.e., 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, pre-key reveal oracles, and state corruption oraclesalso prevent the attacker from knowing all of the above combinations related to any challenge ciphertext at the same time (see Line 22, 35, 71).
  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 event 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

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

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

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

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

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

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

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

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

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

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

```
$\mathbf{sam\text{-}if\text{-}nec}(r)$:
    $\mathsf{flag} \leftarrow \mathsf{bad}$
    $\mathbf{if}\ r = \perp$
        $r \xleftarrow{\$} \mathcal{R}$
        $\mathsf{flag} \leftarrow \mathsf{good}$
    $\mathbf{return}\ (r, \mathsf{flag})$

$\mathbf{record}(\mathtt{P}, \mathsf{type}, \mathsf{flag}, \mathsf{ind}, m, c)$:
    $\mathsf{rec} \leftarrow (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t_{\mathtt{P}}, i_{\mathtt{P}}, m, c)$
    $\mathsf{allTrans}, \mathsf{trans} \xleftarrow{+} \mathsf{rec}$
    $\mathbf{if}\ \neg\mathsf{safe\text{-}inj}_{\neg\mathtt{P}}(t_{\mathtt{P}})$: $\mathsf{comp} \xleftarrow{+} \mathsf{rec}$
    $\mathbf{if}\ \mathsf{type} = \mathsf{chall}$: $\mathsf{allChall}, \mathsf{chall} \xleftarrow{+} \mathsf{rec}$

$\mathbf{ep\text{-}mgmt}(\mathtt{P}, \mathsf{flag}, \mathsf{ind})$:
    $\mathbf{if}\ (\mathtt{P} = \mathtt{A}\ \mathbf{and}\ t_{\mathtt{P}}\ \text{even})\ \mathbf{or}\ (\mathtt{P} = \mathtt{B}\ \mathbf{and}\ t_{\mathtt{P}}\ \text{odd})$
        $\mathbf{if}\ \neg\mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, t_{\mathtt{P}}, \mathsf{ind})$
            $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}} \xleftarrow{+} t_{\mathtt{P}} + 1$
        $t_{\mathtt{P}}{+}{+},\ i_{\mathtt{P}} \leftarrow 0$

$\mathbf{delete}(t, i)$:
    $\mathsf{rec} \leftarrow (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, i, m, c)$ for some $\mathtt{P}, \mathsf{ind}, \mathsf{flag}, m, c$
    $\mathsf{trans}, \mathsf{chall}, \mathsf{comp} \xleftarrow{-} \mathsf{rec}$

$\mathbf{corruption\text{-}update}()$:
    $\mathbf{foreach}\ (\mathtt{P}, \mathsf{ind}, \mathsf{flag}, t, 1, m, c) \in \mathsf{allTrans}$
        $\mathbf{if}\ \neg\mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, (t-1), \mathsf{ind})$
            $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}} \xleftarrow{+} t$
```

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

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

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

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

Figure 3: 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 highlight the difference to the SM-security game for a SM scheme in [1] with blue color.

explained above.

- **Periodic privacy recovery** (PPR)**:** Note that the pre-keys can be periodically generated optionally under fresh randomness. The PPR 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 $\mathtt{A}$) supports a secure environment, such as an HSM. If $\mathtt{A}$'s identity key pair is generated in a secure environment, the private identity key can be neither manipulated nor predicted by any attacker. This means that the attacker can only query $\textsc{NewIdKey}\text{-}\mathtt{A}(r)$ with input $r = \perp$ and never query $\textsc{RevIdKey}\text{-}\mathtt{A}$ 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 $\mathtt{B}$ has access to the fresh randomness, then the privacy of the messages sent from $\mathtt{B}$ to $\mathtt{A}$ always holds.

We stress that our eSM model is strictly stronger than the SM model [1], even without taking the usage of identity keys and pre-keys into account. We provide a detailed comparison in Appendix C for interested readers.

4. Extended Secure Messaging Scheme

In Section 4.1 we describe the intuition behind our eSM construction, followed by a detailed description in Section 4.2. In Section 4.3, we prove the eSM security of our eSM construction and provide concrete instantiations.

4.1. Intuition behind the eSM Construction

Our eSM construction, depicted in Figure 4, uses a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{K.KG}, \mathsf{K.Enc}, \mathsf{K.Dec})$, a digital signature $\mathsf{DS} = (\mathsf{D.KG}, \mathsf{D.Sign}, \mathsf{D.Vrfy})$, a symmetric key encryption $\mathsf{SKE} = (\mathsf{S.Enc}, \mathsf{S.Dec})$, and five key derivation functions KDF_i for $i \in [5]$.

To send a message, the sender runs the KEM encapsulation algorithm three times: the encapsulation upon the partner's latest per-epoch public key, which ensures the privacy against fine-grained state compromise and PCS; the one upon the partner's latest public pre-key, which ensures temporal privacy and the PPR property; and finally the one upon the partner's latest public identity key, which ensures even stronger privacy if the device supports an HSM for storing private identity keys. The sender also signs the outgoing ciphertext using DS and his latest per-epoch signing key to ensure the authenticity against fine-grained state compromise.

Moreover, our eSM construction uses three variants of the NAXOS trick [22], in which ephemeral randomness is combined with a local secret to strengthen against randomness compromise or manipulation attack. First, a symmetric root key $\mathsf{st}.rk$ together with ephemeral randomness is used to derive new shared state when sending the first message in each epoch. This provides strong privacy for new epochs against randomness leakage and manipulation; Second, the sender's local NAXOS string $\mathsf{st}.nxs$ together with the ephemeral randomness is used to improve key generation when sending the first message in each epoch. This provides strong authenticity for the new epoch and strong privacy for the next epoch against randomness leakage and manipulation; Third, the unidirectional ratchet keys *urk* (derived from the shared state) together with the ephemeral randomness are used to derive the real message keys. This ensures FS while preserving immediate decryption with constant-size overhead.

4.2. The eSM Construction in Detail

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 $\{0,1\}^\lambda$. The underlying DS and SKE are assumed to be deterministic. We first introduce the state in our construction.

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

- $\mathsf{st.id}$*: the state owner. In this paper, we have* $\mathsf{st}_{\mathtt{A}}.\mathsf{id} = \mathtt{A}$ *and* $\mathsf{st}_{\mathtt{B}}.\mathsf{id} = \mathtt{B}$.
- $\mathsf{st}.t$*: the local epoch counter. It starts with* 0.
- $\mathsf{st}.i^0$, $\mathsf{st}.i^1$, ...*: the local message index counter of each epoch. They start with* 0.
- $\mathsf{st}.rk \in \{0,1\}^\lambda$*: the (symmetric) root key. This key is initialized from the initial shared secret and updated only when entering next epoch. The root key is used to initialize the chain key at the time of update.*
- $\mathsf{st}.ck^0, \mathsf{st}.ck^1, ... \in \{0,1\}^\lambda$*: the (symmetric) chain keys at each epoch. These keys are initialized at the beginning of each epoch and updated when sending messages. The chain keys are used to deterministically derive the (one-time symmetric) unidirectional ratchet keys (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 (aka. the length) of the transmissions in the previous epochs.*
- $\mathsf{st.prtr}$*: the pre-transcript that is produced 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 (one-time symmetric) unidirectional ratchet keys urk for each epoch. The urks are used to derive the (one-time symmetric) message keys (mk) for real message encryption and decryption using* SKE.
- $(\mathsf{st.ek}^0, \mathsf{st.dk}^0), (\mathsf{st.ek}^1, \mathsf{st.dk}^1), ...$*: the (asymmetric)* KEM *public key pairs. These key pairs are used to encapsulate and decapsulate the randomness, which (together with the unidirectional ratchet key urk) is used to derive the message keys (mk) of* SKE.
- $(\mathsf{st.sk}^{-1}, \mathsf{st.vk}^{-1}), (\mathsf{st.sk}^0, \mathsf{st.vk}^0), (\mathsf{st.sk}^1, \mathsf{st.vk}^1), ...$[3]*: the*

3. The superscript of the signing/verification keys indicates the 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/verify the verification key in epoch 1.

```
IdKGen():
    $(ipk, ik) \xleftarrow{\$} \mathsf{K.KG}()$
    return $(ipk, ik)$
PreKGen():
    $(prepk, prek) \xleftarrow{\$} \mathsf{K.KG}()$
    return $(prepk, prek)$
eInit-A(iss):
    $\mathsf{st_A}.nxs \parallel \_ \parallel \mathsf{st_A}.rk \parallel \mathsf{st_A}.ck^0 \parallel r_{\mathtt{A}}^{\mathsf{KEM}} \parallel r_{\mathtt{B}}^{\mathsf{KEM}} \parallel r_{\mathtt{A}}^{\mathsf{DS}} \parallel r_{\mathtt{B}}^{\mathsf{DS}} \leftarrow iss$
    $(\_, \mathsf{st_A.dk}^0) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{A}}^{\mathsf{KEM}})$, $(\mathsf{st_A.ek}^1, \_) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{B}}^{\mathsf{KEM}})$
    $(\mathsf{st_A.sk}^{-1}, \_) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{A}}^{\mathsf{DS}})$, $(\_, \mathsf{st_A.vk}^0) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{B}}^{\mathsf{DS}})$
    $\mathsf{st_A.id} \leftarrow \mathtt{A}$, $\mathsf{st_A.prtr} \leftarrow \perp$, $\mathsf{st_A}.t \leftarrow 0$, $\mathsf{st_A}.i^0 \leftarrow 0$
    $\mathsf{st_A}.\mathcal{D}_l[\cdot] \leftarrow \perp$, $\mathsf{st_A}.\mathcal{D}_{urk}^0[\cdot] \leftarrow \perp$
    return $\mathsf{st_A}$
eInit-B(iss):
    $\_ \parallel \mathsf{st_B}.nxs \parallel \mathsf{st_B}.rk \parallel \mathsf{st_B}.ck^0 \parallel r_{\mathtt{A}}^{\mathsf{KEM}} \parallel r_{\mathtt{B}}^{\mathsf{KEM}} \parallel r_{\mathtt{A}}^{\mathsf{DS}} \parallel r_{\mathtt{B}}^{\mathsf{DS}} \leftarrow iss$
    $(\mathsf{st_B.ek}^0, \_) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{A}}^{\mathsf{KEM}})$, $(\_, \mathsf{st_B.dk}^1) \xleftarrow{\$} \mathsf{K.KG}(r_{\mathtt{B}}^{\mathsf{KEM}})$
    $(\_, \mathsf{st_B.vk}^{-1}) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{A}}^{\mathsf{DS}})$, $(\mathsf{st_B.sk}^0, \_) \xleftarrow{\$} \mathsf{D.KG}(r_{\mathtt{B}}^{\mathsf{DS}})$
    $\mathsf{st_B.id} \leftarrow \mathtt{B}$, $\mathsf{st_B.prtr} \leftarrow \perp$, $\mathsf{st_B}.t \leftarrow 0$, $\mathsf{st_B}.i^0 \leftarrow 0$, $\mathsf{st_B}.\mathcal{D}_l[\cdot] \leftarrow \perp$
    return $\mathsf{st_B}$
eSend(st, ipk, prepk, m):
    $(c_1, k_1) \xleftarrow{\$} \mathsf{K.Enc}(\mathsf{st.ek}^{\mathsf{st}.t})$, $(c_2, k_2) \xleftarrow{\$} \mathsf{K.Enc}(ipk)$
    $(c_3, k_3) \xleftarrow{\$} \mathsf{K.Enc}(prepk)$
    $(\mathsf{upd}^{\mathsf{ar}}, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$
    if $(\mathsf{st.id} = \mathtt{A}$ and $\mathsf{st}.t\ even)$ or $(\mathsf{st.id} = \mathtt{B}$ and $\mathsf{st}.t\ odd)$
        $l \leftarrow \mathsf{eSend\text{-}Stop}(\mathsf{st})$, $\mathsf{st}.t{+}{+}$, $\mathsf{st}.i^{\mathsf{st}.t} \leftarrow 0$
        $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\mathsf{st}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st}.nxs, r)$
        $(\mathsf{ek}, \mathsf{st.dk}^{\mathsf{st}.t+1}) \xleftarrow{\$} \mathsf{K.KG}(r^{\mathsf{KEM}})$, $(\mathsf{st.sk}^{\mathsf{st}.t}, \mathsf{vk}) \xleftarrow{\$} \mathsf{D.KG}(r^{\mathsf{DS}})$
        $\mathsf{prtr}^{\mathsf{ar}} \leftarrow (l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk})$, $\sigma^{\mathsf{ar}} \leftarrow \mathsf{D.Sign}(\mathsf{st.sk}^{\mathsf{st}.t-2}, \mathsf{prtr}^{\mathsf{ar}})$
        $\mathsf{st.prtr} \leftarrow (\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$, $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$
    $(\mathsf{st}.ck^{\mathsf{st}.t}, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^{\mathsf{st}.t})$, $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})$
    $c' \leftarrow \mathsf{S.Enc}(mk, m)$, $\mathsf{prtr}^{\mathsf{ur}} \leftarrow (\mathsf{st}.t, \mathsf{st}.i^{\mathsf{st}.t}, c', c_1, c_2, c_3)$
    $\sigma^{\mathsf{ur}} \leftarrow \mathsf{D.Sign}(\mathsf{st.sk}^{\mathsf{st}.t}, \mathsf{prtr}^{\mathsf{ur}})$
    return $(\mathsf{st}, (\mathsf{st.prtr}, \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}}))$
eRcv(st, ik, prek, c):
    $((\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}}), \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}}) \leftarrow c$
    $(l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{prtr}^{\mathsf{ar}}$, $(t, i, c', c'_1, c'_2, c'_3) \leftarrow \mathsf{prtr}^{\mathsf{ur}}$
    if $t \le \mathsf{st}.t - 2$: req $\mathsf{st}.\mathcal{D}_l[t] \neq \perp$ and $i \le \mathsf{st}.\mathcal{D}_l[t]$
    req $t \le \mathsf{st}.t + 1$
    req $(\mathsf{st.id} = \mathtt{A}$ and $t\ even)$ or $(\mathsf{st.id} = \mathtt{B}$ and $t\ odd)$
    if $t = \mathsf{st}.t + 1$
        req $\mathsf{D.Vrfy}(\mathsf{st.vk}^{t-2}, \mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$
        $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$, $\mathsf{st}.\mathcal{D}_l[t-2] \leftarrow l$, $\mathsf{st}.t{+}{+}$
        $k_1 \leftarrow \mathsf{K.Dec}(\mathsf{st.dk}^{\mathsf{st}.t}, c_1)$, $k_2 \leftarrow \mathsf{K.Dec}(ik, c_2)$
        $k_3 \leftarrow \mathsf{K.Dec}(prek, c_3)$
        $(\mathsf{upd}^{\mathsf{ar}}, \_) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$
        $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd}^{\mathsf{ar}})$
        $\mathcal{D}_{urk}^{\mathsf{st}.t}[\cdot] \leftarrow \perp$, $\mathsf{st}.i^{\mathsf{st}.t} \leftarrow 0$, $\mathsf{st.ek}^{\mathsf{st}.t+1} \leftarrow \mathsf{ek}$, $\mathsf{st.vk}^{\mathsf{st}.t} \leftarrow \mathsf{vk}$
    req $\mathsf{D.Vrfy}(\mathsf{st.vk}^t, \mathsf{prtr}^{\mathsf{ur}}, \sigma^{\mathsf{ur}})$
    $k'_1 \leftarrow \mathsf{K.Dec}(\mathsf{st.dk}^t, c'_1)$, $k'_2 \leftarrow \mathsf{K.Dec}(ik, c'_2)$
    $k'_3 \leftarrow \mathsf{K.Dec}(prek, c'_3)$
    $(\_, \mathsf{upd}^{\mathsf{ur}}) \leftarrow \mathsf{KDF}_1(k'_1, k'_2, k'_3)$
    while $\mathsf{st}.i^t \le i$
        $(\mathsf{st}.ck^t, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^t)$, $\mathcal{D}_{urk}^t[\mathsf{st}.i^t] \leftarrow urk$, $\mathsf{st}.i^t{+}{+}$
    $urk \leftarrow \mathcal{D}_{urk}^t[i]$, $\mathcal{D}_{urk}^t[i] \leftarrow \perp$, req $urk \neq \perp$
    $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd}^{\mathsf{ur}})$ , $m \leftarrow \mathsf{S.Dec}(mk, c')$
    return $(\mathsf{st}, t, i, m)$
```

Figure 4: Our eSM construction. $\mathsf{KEM} = (\mathsf{K.KG}, \mathsf{K.Enc}, \mathsf{K.Dec})$, $\mathsf{DS} = (\mathsf{D.KG}, \mathsf{D.Sign}, \mathsf{D.Vrfy})$, and $\mathsf{SKE} = (\mathsf{S.Enc}, \mathsf{S.Enc})$ respectively denote a key encapsulation mechanism, a deterministic digital signature and a deterministic authenticated encryption schemes. The KDF_i for $i \in [5]$ denote five independent key derivation functions.

(asymmetric) DS *private key pairs, which are used to sign and verify the (new) pre-transcript output by* eSend.

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

- eRcv-Max(st, l): This algorithm is called in eRcv algorithm when the caller switches its role from message sender in epoch $\mathsf{st}.t$ to 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. Once l messages corresponds to the old epoch t' are received, the state values for receiving messages in epoch t', i.e., $\mathsf{st}.i^{t'}$, $\mathsf{st}.ck^{t'}$, $\mathsf{st.dk}^{t'}$, $\mathsf{st.vk}^{t'}$, $\mathsf{st}.\mathcal{D}_{urk}^{t'}$, $\mathsf{st}.\mathcal{D}_l[t']$ are erased, i.e., set to $\perp$. Moreover, the number how many times the chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ has been forwarded (i.e., how many messages have been sent) in the epoch $\mathsf{st}.t$ is stored, while the chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ itself together with the encryption key $\mathsf{st.ek}^{\mathsf{st}.t}$ is erased.
- 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 epoch $\mathsf{st}.t - 1$, which is locally stored during the previous eRcv-Max invocation, denoted by l. The signing key $\mathsf{st.sk}^t$ is also erased after its signs the next verification key $\mathsf{st.vk}^{t+2}$ later. We write $l \leftarrow \mathsf{eSend\text{-}Stop}(\mathsf{st})$.

Following the syntax in Definition 1, our eSM construction consists of following six algorithms below.

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

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

eInit-A(iss)*:* The A's extended initialization algorithm inputs an initial shared secret $iss \in \mathcal{ISS}$. First, A parses iss into seven components: the initial NAXOS string $\mathsf{st_A}.nxs$, the shared root key $\mathsf{st_A}.rk$, the shared chain key $\mathsf{st_A}.ck^0$, and four randomness for A's and B's KEM and DS key generation: $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, $r_{\mathtt{B}}^{\mathsf{DS}}$. Then, A respectively runs K.KG and D.KG on the above randomness and stores $\mathsf{st_A.dk}^0$, $\mathsf{st_A.ek}^1$, $\mathsf{st_A.sk}^{-1}$, $\mathsf{st_A.vk}^0$, 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, A sets the identity $\mathsf{st_A.id}$ to A, the local pre-transcript $\mathsf{st_A.prtr}$ to $\perp$, the epoch counter $\mathsf{st_A}.t$ to 0, the message index $\mathsf{st_A}.i^0$ 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 $\mathsf{st_A}$.

eInit-B(iss)*:* The B's extended initialization algorithm inputs an initial shared secret $iss \in \mathcal{ISS}$ and runs very similar to eInit-A. First, B parses iss into seven components: the initial NAXOS string $\mathsf{st_B}.nxs$, the shared root key $\mathsf{st_B}.rk$, the shared chain key $\mathsf{st_B}.ck^0$, and four randomness for A's and B's KEM and DS key generation: $r_{\mathtt{A}}^{\mathsf{KEM}}$, $r_{\mathtt{B}}^{\mathsf{KEM}}$, $r_{\mathtt{A}}^{\mathsf{DS}}$, $r_{\mathtt{B}}^{\mathsf{DS}}$. Then, B respectively runs K.KG and D.KG on the above randomness and stores $\mathsf{st_B.ek}^0$, $\mathsf{st_B.dk}^1$, $\mathsf{st_B.vk}^{-1}$, $\mathsf{st_A.sk}^0$, 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 B is the ones discarded by A, and vice versa.

Finally, B sets the identity $\mathsf{st_B.id}$ to B, the local pre-transcript $\mathsf{st_B.prtr}$ to $\perp$, the epoch counter $\mathsf{st_B}.t$ to 0, the message index $\mathsf{st_B}.i^0$ to 0, and initializes the maximal transmission length dictionary $\mathcal{D}_l$, followed by outputting the state $\mathsf{st_B}$. Note that no unidirectional ratchet dictionary $\mathcal{D}^0_{urk}$ is initialized, since 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 st.id is A and the epoch $\mathsf{st_A}.t$ is even or the caller is B and the epoch is odd, it first executes the following so-called *asymmetric ratchet* (ar) framework: First, the caller runs eSend-Stop(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 $\mathsf{st}.i^{\mathsf{st}.t}$ 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 signs the pre-transcript for the ar framework $\mathsf{prtr}^{\mathsf{ar}}$, including the value l, the ciphertext c_1, c_2, and c_3, the newly sampled encapsulation key ek and the verification key vk, using the signing key produced two epochs earlier $\mathsf{st.sk}^{\mathsf{st}.t-2}$ for a signature σ^{ar}. The pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ and signature σ^{ar} are stored into the state st.prtr. Finally, the caller forwards the ar framework 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 $\mathsf{st}.ck^{\mathsf{st}.t}$.

Next, the caller executes the so-called *unidirectional ratchet* (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 $c' \leftarrow \mathsf{S.Enc}(mk, m)$. Finally, the caller signs the pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$ of the ur framework, including the epoch $\mathsf{st}.t$, the message index $\mathsf{st}.i^{\mathsf{st}.t}$, and the ciphertexts c', c_1, c_2, and c_3, for a signature σ^{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 and signature $\mathsf{st.prtr} = (\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$, the ur pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$, and the signature σ^{ur}.

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

First, the caller parses the input ciphertext c into the pre-transcript and signature of ar framework $(\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$, the unidirectional ratchet pre-transcript $\mathsf{prtr}^{\mathsf{ur}}$, and the signature σ^{ur}. Next, the caller further parses the pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ into one number l, three ciphertexts c_1, c_2, and c_3, an encapsulation key ek, and a verification key vk, and parses $\mathsf{prtr}^{\mathsf{ur}}$ 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 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 $m = \perp$.

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 pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ 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, sets the message counter $\mathsf{st}.i^{\mathsf{st}.t}$ to 0, and locally stores the encapsulation key for the next epoch and verification key for this epoch.

Then, the caller executes the unidirectional ratchet framework, no matter whether ar 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-transcript $\mathsf{prtr}^{\mathsf{ur}}$. 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 keys into the dictionary and incrementing the message index by 1, until the local message index $\mathsf{st}.i^t$ reaches the parsed message index i. In the end, the caller reads the unidirectional ratchet key urk from the dictionary corresponding to the parsed message index, followed by erasing it from the dictionary. It must hold that $urk = \perp$ and aborts otherwise. The caller then derives the message key mk by applying 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 and message index i, and the decrypted message m.

4.3. Security Conclusion and Concrete Instantiation

Theorem 1. *Let Π denote our* eSM *construction in Section 4.2. If the underlying* KEM *is δ_{KEM}-strongly correct*[4] *and $\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}$-secure,* DS *is δ_{DS}-strongly correct and $\epsilon^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}$-secure,* SKE *is δ_{SKE}-strongly correct and $\epsilon^{\mathsf{IND\text{-}1CCA}}_{\mathsf{SKE}}$-secure,* KDF_1 *is $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$-secure*[5]*,* KDF_2 *is $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$ secure,* KDF_3 *is $\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}$-secure,* KDF_4 *is $\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4}$-secure,* KDF_5 *is $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$-secure, in time t, then Π is $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-*eSM *secure for $\triangle_{\mathsf{eSM}} = 2$, where*

$$\begin{aligned}\epsilon \le\ & (q_{\mathsf{ep}} + q)\delta_{\mathsf{DS}} + 3(q_{\mathsf{ep}} + q)\delta_{\mathsf{KEM}} + q\delta_{\mathsf{SKE}} + q_{\mathsf{ep}}\epsilon^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}\\ &+ q^2_{\mathsf{ep}}q_{\mathsf{M}}(q+1)\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + q_{\mathsf{ep}}(q_{\mathsf{M}}+2)q\epsilon^{\mathsf{IND\text{-}1CCA}}_{\mathsf{SKE}}\\ &+ q^2_{\mathsf{ep}}q_{\mathsf{M}}(q+1)\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + q^2_{\mathsf{ep}}(q_{\mathsf{ep}}q + q_{\mathsf{ep}} + 1)\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}\\ &+ q^2_{\mathsf{ep}}(q+1)\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3} + q_{\mathsf{ep}}q(q+1)\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4}\\ &+ q_{\mathsf{ep}}(q_{\mathsf{ep}}q_{\mathsf{M}}q + q_{\mathsf{ep}}q_{\mathsf{M}} + 2q)\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}\end{aligned}$$

Proof. Our proof is divided into two steps: First, we modularize the eSM security into three simplified security notations: correctness, privacy, and authenticity, which are defined in Appendix G.

Second, we introduce four lemmas in Appendix H.1. Lemma 1 reduces the eSM security to the simplified security notions, the full proof of which is given in Appendix H.2. Lemma 2, 3, and 4 respectively proves the simplified correctness, privacy, and authenticity of our eSM construction in Section 4.2, the full proof of which are given in Appendix H.3, H.4, and H.5. The proof is concluded by combining the above four lemmas together. □

Instantiation: We give the concrete instantiation for both classical and PQ settings. The deterministic DS can be instantiated with Ed25519 for classical setting, the formal analysis was given in [23], and the NIST suggested CRYSTALS-Dilithium for the PQ security, which is analyzed in [24]. A generic approach to instantiating KEM is to encrypt random strings using deterministic OW-CCA or merely OW-CPA secure PKE for strong correctness [25], [26]. The NIST suggested NTRU is also available for IND-CCA security and strong correctness [27]. The deterministic IND-1CCA secure authenticated encryption SKE can be instantiated with the Encrypt-then-MAC construction in [28]. 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, as explained in Appendix F.4. We suggest to double the security parameter of the symmetric primitives for PQ security.

4. By strongly correct, we mean that the schemes are conventionally correct for all randomness. See Appendix F.

5. By 3prf security, we mean that a function is indistinguishable from a random function w.r.t any of the three inputs. See Appendix F.4.

5. Offline Deniability

As explained in Section 2.3, although the combinations of SPQR and ACD19 or our eSM achieve strong privacy and authenticity in the PQ setting, it is still an open question what flavors of offline deniability can be achieved by the combined protocols in the PQ setting. To address this, we extend the game-based offline deniability for asynchronous DAKE scheme Σ [11] to its combination with an eSM scheme Π.

Our offline deniability experiment is depicted in Figure 5. 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 overline. The difference to the original model in [11, Definition 11], also see Definition 7 in Appendix E, is highlighted with blue color.

The experiment 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, long-term identity and medium-term pre- public/private key pairs of Σ and Π are generated for all honest parties and provided to the attacker (e.g., the judge). A challenge bit $\mathsf{b} \in \{0, 1\}$ is randomly sampled. The attacker (i.e., the judge) is given repeated access to two oracles below and wins if it can distinguish any real conversation transcript (i.e., $\mathsf{b} = 0$) from a fake transcript produced by Fake algorithms that simulate any accuser's view (i.e., $\mathsf{b} = 1$). This means, if a protocol is offline deniable, then the judge cannot decide whether a transcript given by the accuser is the real transcript of the conversation with the defendant or produced by the accuser alone.

Oracle $\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$***:*** This oracle inputs are a sender identity sid, a receiver identity rid, an 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, the session counter n is incremented by 1 and the set of the sender identity sid and the receiver identity rid is set to $\mathcal{D}_{\mathsf{session}}[i]$. Then, it simulates the honest DAKE 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 $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, \overline{ipk}_{\mathsf{did}}, \overline{ik}_{\mathsf{aid}}, \mathcal{L}^{prek}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ to produce a fake state $\mathsf{st}^n_{\mathsf{Fake}}$. The transcript T is returned.

Oracle $\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$***:*** This oracle inputs a sender identity sid, a receiver identity rid, a session index i, 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. 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

```
$\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$:
1   $\mathcal{D}_{\mathsf{session}}[\cdot] \leftarrow \bot,\ n \leftarrow 0$
2   $\mathcal{L}_{\mathsf{all}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \leftarrow \emptyset$
3   for $u \in [q_{\mathsf{P}}]$
4       $\mathcal{L}^{\overline{prek}}_{u} \leftarrow \emptyset$
5       $\mathcal{L}^{prek}_{u} \leftarrow \emptyset$
6       $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$
7       $(ipk_u, ik_u) \xleftarrow{\$} \Pi.\mathsf{IdKGen}()$
8       $\mathcal{L}^{\overline{ipk}}_{\mathsf{all}} \xleftarrow{+} \{\overline{ipk}_u\}$
9       $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{ipk}_u, \overline{ik}_u)$
10      $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (ipk_u, ik_u)$
11      for $\mathsf{ind} \in [q_{\mathsf{M}}]$
12          $(\overline{prepk}^{\mathsf{ind}}_u, \overline{prek}^{\mathsf{ind}}_u) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$
13          $(prepk^{\mathsf{ind}}_u, prek^{\mathsf{ind}}_u) \xleftarrow{\$} \Pi.\mathsf{PreKGen}()$
14          $\mathcal{L}^{\overline{prek}}_{u} \xleftarrow{+} \overline{prek}^{\mathsf{ind}}_u,\ \mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \xleftarrow{+} \overline{prepk}^{\mathsf{ind}}_u$
15          $\mathcal{L}^{prek}_{u} \xleftarrow{+} prek^{\mathsf{ind}}_u$
16          $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{prepk}_u, \overline{prek}_u)$
17          $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (prepk_u, prek_u)$
18  $\mathsf{b} \xleftarrow{\$} \{0,1\}$
19  $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathcal{L}_{\mathsf{all}})$
20  return $[\![\mathsf{b} = \mathsf{b}']\!]$
```

```
$\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did}, \mathsf{ind})$:
21  req $\{\mathsf{aid}, \mathsf{did}\} = \{\mathsf{sid}, \mathsf{rid}\}$ and $\mathsf{sid} \neq \mathsf{rid}$
22  $n{+}{+},\ \mathcal{D}_{\mathsf{session}}[n] \leftarrow \{\mathsf{sid}, \mathsf{rid}\}$
23  if $\mathsf{b} = 0$ or $\mathsf{aid} = \mathsf{sid}$
24      $\pi_{\mathsf{rid}}.role \leftarrow \mathtt{resp},\ \pi_{\mathsf{rid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
25      $\pi_{\mathsf{sid}}.role \leftarrow \mathtt{init},\ \pi_{\mathsf{sid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
26      $(\pi'_{\mathsf{rid}}, m) \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{rid}}, (\mathsf{create}, \mathsf{ind}))$
27      $(\pi'_{\mathsf{sid}}, m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{sid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{sid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{sid}}, m)$
28      $(K, T) \xleftarrow{\$} (\pi'_{\mathsf{sid}}.K, (m, m'))$
29  else
30      $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_{\mathsf{sid}}, \overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathsf{ind})$
31  if $\mathsf{b} = 0$
32      $\mathsf{st}^{n}_{\mathsf{sid}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}B}(K),\ \mathsf{st}^{n}_{\mathsf{rid}} \xleftarrow{\$} \Pi.\mathsf{eInit\text{-}A}(K)$
33  else
34      $\mathsf{st}^{n}_{\mathsf{Fake}} \xleftarrow{\$} \mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}^{prek}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$
35  return $T$
$\mathsf{Session\text{-}Execute}(\mathsf{sid}, \mathsf{rid}, i, \mathsf{ind}, m)$:
36  req $\mathcal{D}_{\mathsf{session}}[i] = \{\mathsf{sid}, \mathsf{rid}\}$
37  if $\mathsf{b} = 0$
38      $(\mathsf{st}^{i}_{\mathsf{sid}}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}^{i}_{\mathsf{sid}}, ipk_{\mathsf{rid}}, prepk^{\mathsf{ind}}_{\mathsf{rid}}, m)$
39      $(\mathsf{st}^{i}_{\mathsf{rid}}, \_, \_, \_) \leftarrow \Pi.\mathsf{eRcv}(\mathsf{st}^{i}_{\mathsf{rid}}, ik_{\mathsf{rid}}, prek^{\mathsf{ind}}_{\mathsf{rid}}, c)$
40  else
41      $(\mathsf{st}^{i}_{\mathsf{Fake}}, c) \xleftarrow{\$} \mathsf{Fake}^{\mathsf{eSend}}_{\Pi}(\mathsf{st}^{i}_{\mathsf{Fake}}, ipk_{\mathsf{rid}}, prepk^{\mathsf{ind}}_{\mathsf{rid}}, m, \mathsf{sid}, \mathsf{rid}, \mathsf{ind})$
42  return $c$
```

Figure 5: The offline deniability experiment for an attacker $\mathcal{A}$ against the combination of a DAKE scheme Σ and an eSM scheme Π. The experiment $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}$ is parameterized the maximal numbers of parties q_{P}, pre-keys per party q_{M}, and total sessions q_{S}. We highlight the difference to the offline deniability experiment for DAKE in Definition 7 with blue color.

receiver identity rid, and a pre-key index ind. In both cases, the ciphertext c is returned.

We stress that our offline deniability model is a significant extension to the one for DAKE in [11]. First, our model also allows the attacker (e.g. the judge) to obtain *all* initial private secret of all parties, as in [11].

Second, while the model in [11] prevents an attacker from deciding the challenge bit b given the (output) shared keys and the transcripts of DAKE key establishments, our model prevents an attacker from deciding b given the transcripts of full conversations, which include the one of DAKE and the one of eSM inputting the shared key of DAKE. This extension follows the idea behind the simulation-based extension [13].

Third, the accuser in the model for DAKE in [11] must play the role of a responder `resp` (i.e., the receiver rid during the key establishment) rather than an initiator (i.e., the sender sid during the key establishment), since the Σ.Fake algorithm is only defined on the responder's behalf. The main reason behind is that all transcripts in a DAKE scheme are produced by the initiator alone. However, the responder producing no output during the key establishment might produce some transcripts afterwards. To capture this, our model also allows the accuser to be the initiator `init` in the whole conversation. In fact, our Session-Execute simulates the accuser's view (when $\mathsf{b} = 1$) by running the Fake_{Π} algorithm that simulates the stateful execution of either the initiator or the responder, depending on whether $\mathsf{aid} = \mathsf{sid}$ or rid in the corresponding Session-Start query[6].

6. In our model, we restrict the behavior of the accuser, who acts as initiator, to be honest during the key establishment phase, see Line 23. We leave a stronger model without this restriction as future work.

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

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

where q_{P}, q_{M}, *and* q_{S} *respectively denote the maximal number of parties, of pre-key per party, and total sessions in the* $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}$ *in Figure 5.*

Theorem 2. *Let* Σ *denote a* DAKE *scheme and* Π *denote our* eSM *construction in Section 4.2. If* Σ *is* (t, ϵ, q)*-deniable (with respect to any* q_{P}, q_{M}*) in terms of the Definition 7 , then the composition of* Σ *and* Π *is* $(t, \epsilon, q_{\mathsf{P}}, q_{\mathsf{M}}, q)$*-deniable.*

Proof Sketch. We define $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}$ algorithm as running both eInit-A and eInit-B upon the input K and storing all other inputs. We define $\mathsf{Fake}^{\mathsf{eSend}}_{\Pi}$ algorithm as honest execution of Π.eSend upon sender sid followed by Π.eRcv upon the receiver rid and the ciphertext of Π.eSend. If the attacker cannot distinguish the real DAKE transcripts and output keys from the fake ones, then it cannot distinguish the real DAKE and eSM (and therefore the full) transcripts from the fake ones. We give the full proof in Appendix H.6. □

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, pp. 129–158.

[2] J. Pijnenburg and B. Poettering, "On Secure Ratcheting with Immediate Decryption," in *Advances in Cryptology – ASIACRYPT 2022*. Cham: Springer Nature Switzerland, 2022, pp. 89–118.

[3] M. Marlinspike and T. Perrin, "The X3DH Key Agreement Protocol," November 2016, https://signal.org/docs/specifications/x3dh/.

[4] T. Perrin and M. Marlinspike, "The Double Ratchet Algorithm," November 2016, https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[5] 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, pp. 451–466.

[6] 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, pp. 33–62.

[7] B. Poettering and P. Rösler, "Towards Bidirectional Ratcheted Key Exchange," in *Advances in Cryptology – CRYPTO 2018*. Springer, 2018, pp. 3–32.

[8] D. Jost, U. Maurer, and M. Mularczyk, "Efficient Ratcheting: Almost-Optimal Guarantees for Secure Messaging," in *Advances in Cryptology – EUROCRYPT 2019*. Springer, 2019, pp. 159–188.

[9] F. B. Durak and S. Vaudenay, "Bidirectional Asynchronous Ratcheted Key Agreement with Linear Complexity," in *Advances in Information and Computer Security*. Springer, 2019, pp. 343–362.

[10] A. Bienstock, J. Fairoze, S. Garg, P. Mukherjee, and S. Raghuraman, "A More Complete Analysis of the Signal Double Ratchet Algorithm," Cryptology ePrint Archive, Paper 2022/355, https://eprint.iacr.org/2022/355. [Online]. Available: https://eprint.iacr.org/2022/355

[11] 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, 2021, https://ia.cr/2021/769.

[12] 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, pp. 1914–1983, 2020. [Online]. Available: https://doi.org/10.1007/s00145-020-09360-1

[13] 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, pp. 188–209.

[14] F. Balli, P. Rösler, and S. Vaudenay, "Determining the core primitive for optimally secure ratcheting," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 621–650.

[15] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs, "Ratcheted encryption and key exchange: The security of messaging," in *Annual International Cryptology Conference*. Springer, 2017, pp. 619–650.

[16] M. Di Raimondo, R. Gennaro, and H. Krawczyk, "Deniable Authentication and Key Exchange," in *ACM CCS*. ACM, 2006, p. 400–409. [Online]. Available: https://doi.org/10.1145/1180405.1180454

[17] N. Unger and I. Goldberg, "Deniable Key Exchanges for Secure Messaging," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, p. 1211–1223. [Online]. Available: https://doi.org/10.1145/2810103.2813616

[18] ——, "Improved strongly deniable authenticated key exchanges for secure messaging," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 1, pp. 21–66, 2018.

[19] J. Brendel, M. Fischlin, F. Günther, C. Janson, and D. Stebila, "Towards post-quantum security for Signal's X3DH handshake," in *International Conference on Selected Areas in Cryptography*. Springer, 2020, pp. 404–430.

[20] S. Dobson and S. D. Galbraith, "Post-Quantum Signal Key Agreement with SIDH," Cryptology ePrint Archive, 2021, https://ia.cr/2021/1187.

[21] 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.

[22] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger Security of Authenticated Key Exchange," in *Provable Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–16.

[23] 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 (S&P)*. IEEE, 2021, pp. 1659–1676.

[24] 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.

[25] 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, pp. 360–378.

[26] D. J. Bernstein and E. Persichetti, "Towards KEM Unification," Cryptology ePrint Archive, Paper 2018/526, https://eprint.iacr.org/2018/526. [Online]. Available: https://eprint.iacr.org/2018/526

[27] 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.

[28] 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, pp. 531–545.

[29] M. Bellare and A. Lysyanskaya, "Symmetric and Dual PRFs from Standard Assumptions: A Generic Validation of an HMAC Assumption," Cryptology ePrint Archive, Report 2015/1198, 2015, https://ia.cr/2015/1198.

Appendix A. Review of ACD19 and TR protocols

The* ACD19 *protocol [1, Section 5.1]: The ACD19 protocol is an instance of the SM scheme and can be further modularized into three building blocks: the *Continuous Key Agreement* (CKA), where the sender exchanges its randomness with the partner; 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 refreshes its inherent shared state by using the randomness of provided by CKA and initializes a new FS-AEAD thread, which provides the post-compromise security.

The ACD19 protocol is managed according to the epoch, which is used to describe how many interactions in a two-party communication channel have been processed. The behavior of a party (assume A) for sending messages is different when A enters a new epoch or not:

1) When a receiver A switches to sender and sends the first message in a new epoch, A first counts and remembers how many messages have been sent in the last epoch using the corresponding FS-AEAD thread, which is then erased. Next, A increments the inherent epoch counter by 1. Then, A invokes the sending algorithm of the CKA component

for exchanging the randomness with the partner B. The output of CKA algorithm in this epoch is also remembered locally. Afterwards, A refreshes the shared state using PRF-PRNG and initialize a new FS-AEAD thread for the new epoch.
2) Regardless of whether A is sending the first message in a new epoch (after executing the above step) or sending subsequent messages in the current epoch, A uses the current FS-AEAD thread for the encrypting real message with associated data: the number of messages sent two epoch earlier, the output of CKA in this epoch, the current epoch counter.

The receiving process is defined in the reverse way. When a sender (assume B) receives a message indicating the next epoch, B switches his role to receiver and enters the next epoch by incrementing the internal epoch counter. Notably, B parses and locally remembers the number of messages sent two epochs earlier from the received ciphertext and erases the FS-AEAD thread once these messages arrived at B.

Moreover, several different instantiations of CKA, FS-AEAD, and PRF-PRNG components are also given in [1].

The* TR *protocol [10, Section 5.1]: The Triple Ratchet (TR) is very close to the ACD19 construction in [1], except for the following two differences:

1) When a party switches its role from receiver to sender, it does not count and remember how many messages have been sent in the last epoch. Instead, this step is executed in the receiving algorithm when a party enters a new epoch and switches its role from sender to receiver.
2) The underlying CKA component must be instantiated with a customized CKA+ construction, which provides better privacy against randomness leakage but relies on a non-standard assumption and a random oracle. Note that CKA is a generic building block, while CKA+ is a concrete instantiation. The other building blocks such as FS-AEAD and PRF-PRNG can be instantiated with the constructions in [1].

For the interested readers, we also compare ACD19 and TR with our protocol in Appendix D.

Appendix B. Review on messaging protocols with various optimal security

The "optimal" protocols by Jäger and Stepanovs [6] and by Pöttering Rösler [7], the "sub-optimal" protocol by Durak and Vaudenay [9], and a novel protocol by Pijnenburg and Pöttering [2] (we call "ID-optimal"), all are post-quantum compatible. The "almost-optimal" protocol by Jost, Maurer, and Mularczyk [8] only has classically secure instantiation. Technically, they follow different ratcheting frameworks:

(1) "optimal" Jäger-Stepanovs protocol [6]: 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 [7]: 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 [9]: 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.

(4) "almost-optimal" Jost-Maurer-Mularczyk protocol [8]: The Jost-Maurer-Mularczyk protocol aims at stronger security than what is achieved by Signal, but slightly weaker than optimal security proposed in Jäger-Stepanovs' and Pöttering-Rösler's work, yet its efficiency is closer to that of Signal. The Jost-Maurer-Mularczyk protocol employs two customized novel schemes: healable and key-updating encryption (HkuPke) and key-updating signatures (KuSig). When Alice sends messages to Bob, Alice first samples two DS key pairs, while the one is used by Alice for sending next continuous message, the other is used by Bob for sending the reply. Next, Alice updates the key of HkuPke and encrypts the message as well as the private DS signing key for Bob. Then, Alice signs the transcript and her next DS verification key twice, by using KuSig and DS. Finally, the state is updated. Note that the sender has to send the next DS signing and verification keys to the partner. If one message is lost in the transmission (from either Alice or Bob), the receiver can neither verify the next message from the partner nor send a valid reply to the partner – the communication session becomes stuck.

Moreover, Jost-Maurer-Mularczyk's HkuPke construction uses a customized secretly key-updatable encryption (SkuPke), the only known instantiation of which relies on the Diffie-Hellman exchange, for which currently no PQ-secure instantiation is available.

(5) "ID-optimal" Pijnenburg-Pöttering protocol [2]: The Pijnenburg-Pöttering protocol aims to solve the weak forward secrecy caused by the immediate decryption by definition. In principle, the immediate decryption requires every receiver to be able to decrypt a ciphertext at the time of arrival. Thus, if an attacker can intercept a message and corrupt the receiver's state in the future, the attacker can always recover the plaintext from the intercepted ciphertext.

To solve this, the Pijnenburg-Poettering protocol employs three updatable mechanisms: Updatable Signature Schemes (USS), Key-Evolving KEM (KeKEM), and Key-Updatable KEM (KuKEM). Unlike all above protocols, while keys of the KuKEM and USS schemes are updated whenever a party switches the role from receiver to sender, the keys of KeKEM are updated every certain time interval. If a past message does not arrive at the receiver, the receiver still stores the corresponding decryption keys for the decryption at the time of message arrival, however, but only within a fixed length of time. After a pre-defined time interval, the corresponding decryption keys are expired and cleaned from the local state. By this, the compromise of a party's state does not cause the message leakage that is sent long time ago.

In particular, none of these protocols provide immediate decryption with constant-size overhead.

Appendix C. Security Model Comparison between our eSM and SM in [1]

Our eSM model extends the SM model [1], with the following main differences.

Extended Syntax. Compared to the original SM definition [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 generated identity and pre-key pairs both are used in the eSend and eRcv algorithms for sending and receiving messages.

More Expected Security Properties. Our eSM is expected to preserve all basic properties of the SM schemes in [1], including correctness, immediate decryption, FS, and PCS. Moreover, our eSM targets the stronger authenticity and privacy than SM in [1]. In particular, the authenticity and privacy in [1] hold only when neither parties' states are compromised. Instead, we aim for stronger authenticity and privacy against more fine-grained state compromise. This potentially indicates that our eSM achieves stronger randomness leakage/failures property. Finally, our eSM also aims at two new properties: state compromise/failures and PPR, which are not captured by SM in [1].

Stronger Security Model. Our eSM model is more complicated than the SM model [1] from many aspects. First, our eSM model needs more variables that are related to the identity keys and pre-keys, which are excluded 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 allTrans and allChall to simplify the security analysis of the benefits obtained from using the identity keys and pre-keys. Besides, 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.

Second, we define two new safe predicates $\mathsf{safe}_{\mathtt{P}}^{\mathsf{preK}}$ and $\mathsf{safe\text{-}st}_{\mathtt{P}}$, which respectively capture the safety of the the pre-key and session state. The $\mathsf{safe\text{-}ch}_{\mathtt{P}}$ and $\mathsf{safe\text{-}inj}_{\mathtt{P}}$ predicates were introduced in [1]. However, our eSM model defines them in a different way: 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 $\mathsf{safe\text{-}ch}_{\mathtt{P}}$ predicate in [1] equals the condition (a), our new conditions (b), (c), and (d) respectively capture the strong privacy, state compromise/failures, and PPR security properties. Moreover, our $\mathsf{safe\text{-}inj}_{\mathtt{P}}$ additionally inputs an epoch number t.

We stress that 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.

Third, our eSM model has one new helper function **corruption-update**. The other four helper functions in our eSM model are introduced in [1], but are defined with slight differences due to our new notations.

Finally, our eSM model includes 8 new oracles that are not included in SM [1]. The new oracles are related to the identity keys and pre-keys. Besides, the other 8 oracles for message transmissions are identical to the one in SM model, except for the notation differences. The only oracles that have huge differences with the ones in SM model are state corruption oracles: While our corrupt oracles requires any of three conditions holds: (1) the chall does not include the record produced by the partner $\neg\mathtt{P}$, (2) the flag in the record is good and $\mathtt{P}$'s identity key is safe, and (3) the flag in the record is good and $\mathtt{P}$'s pre-key corresponding to the pre-key index in the record is safe, the ones in SM model only require the condition (1). After that, the corruption oracle in our eSM model adds all records rec ∈ trans, which are produced by $\neg$P at an unsafe epoch t (but not all epochs as in [1]), into the compromise set comp.

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 for more epochs.

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

Appendix D. Comparison of our eSM construction with ACD19 and TR

Although our eSM construction in Section 4.2, the ACD19 in [1], and the TR construction in [10], all satisfy immediate decryption with constant bandwidth consumption, their designs differ in many details.

***Comparison between our* eSM *construction and* ACD19:** The ACD19 protocol in [1, Section 5.1] employs 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 sample 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 obvious 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 indicate the compromise of the decapsulation key (resp. signing key) and preserves the privacy (resp. authenticity).

Second, our construction makes use of the identity keys and pre-keys, which also provide benefits in terms of strong privacy, state compromise/failure, and PPR. If the corruption of a device's full state without secure environment is not noticed by the owner (which is the common real-world scenario), the privacy for subsequent messages from the partner is lost until the corruption party sends a reply. The use of pre-key provides mitigation in this scenario as the pre-key is updated every certain period in the back-end without the active behavior of the corrupted party. Moreover, if the device has a secure environment such as an HSM, storing identity keys into the HSM provides even stronger security guarantees, as we explained in Section 3.3.

Finally, our construction implicitly uses three kinds of NAXOS-like tricks for strong privacy. (1) First, the symmetric root key together with ephemeral randomness is used for deriving new shared state when sending the first message in each epoch, this is same as in ACD19. (2) Second, the NAXOS string st.nxs (in the sender's state) together with the ephemeral randomness is used for improving the key generation when sending the first message in each epoch. (3) Third, the unidirectional ratchet keys (derived from the shared state) together with the ephemeral randomness are used to derive the real message keys. We stress the second and third NAXOS tricks provide additional benefits to our construction when comparing with ACD19. On the one hand, bad randomness quality of a party when sending the first message in a new epoch will cause leakage of the private KEM key in ACD19, but not in our construction. In this case, the corruption of the partner in the next epoch will cause the loss of privacy in ACD19, but not in our construction, due to the second NAXOS trick. On the other hand, the message keys are derived from not only the mere state but also ephemeral randomness. The third NAXOS trick together with the usage of identity keys and pre-keys provide stronger privacy against state corruption attacks.

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 4.3.

***Comparison between our* eSM *construction and* TR:** The TR construction in [10, Section 5.1] is very close to the one in ACD19 except for two differences: (1) The FS-Stop function of the underlying FS-AEAD components is invoked when receiving the first message in a new epoch but not sending. (2) The underlying CKA component must be instantiated with a new customized CKA+ construction based on a Diffie-Hellman exchange. The state of CKA+ component does not merely rely on the randomness but also on the past state. This can be seen as a variant of the NAXOS trick.

Compared to the TR construction, our eSM construction mainly differs in four aspects: First, our construction employs generic KEMs aiming at post-quantum compatibility, while TR makes use of a concrete Diffie-Hellman exchange, which is vulnerable to quantum attacks.

Second, while TR and our constructions both use the root key for a NAXOS trick, the NAXOS trick for improving privacy of the KEM key pairs is different. While TR uses a tailored CKA+ construction assuming a non-standard StDH and random oracles, our construction uses a local NAXOS string $\mathsf{st}.nxs$ only assuming the dual security of the function KDF_2, the generic constructions of which based solely on standard assumptions are given in [29].

Third, TR and our construction both prevent an attacker from corrupting the receiver in the current epoch and forging a ciphertext corresponding to the previous epoch to the partner by erasing a party's state for sending messages once a message from the partner for the next epoch arrives. Note that this attack is effective against ACD19, as the attacker can in the current epoch corrupt the FS-AEAD thread corresponding to the previous epoch and use it to encrypt the forged message. The only difference Due to the immediate decryption property, the forged ciphertext must be correctly decrypted. The TR construction prevents this attack by invoking FS-Stop function when receiving the first message in a new epoch to erase the chain key for sending in the previous epoch. In contrast, our construction prevents this attack by erasing both the chain key and the KEM encapsulation key for sending in the old epoch in the eRcv-Max function.

Fourth, the remaining benefits of our construction in comparison to ACD19 also apply to the comparison with TR, including strong privacy, strong authenticity, PPR, the resilience to a novel forgery attack.

Appendix E. Review on DAKE scheme and the game-based deniability

We recall the DAKE scheme and its offline deniability notion from [11].

E.1. The DAKE scheme

Definition 6. *An asynchronous deniable authenticated key exchange (*DAKE*) protocol Σ is a tuple of algorithms $\Sigma = (\Sigma.\mathsf{IdKGen}, \Sigma.\mathsf{PreKGen}, \Sigma.\mathsf{EpKGen}, \Sigma.\mathsf{Run}, \Sigma.\mathsf{Fake})$ as defined below.*

- ***(Long-term) identity key generation*** $(\overline{ipk}_u, \overline{ik}_u) \overset{\$}{\leftarrow} \Sigma.\mathsf{IdKGen}()$*: outputs the identity public/private key pair of a party u.*
- ***(Medium-term) pre-key generation*** $(\overline{prepk}_u^{\mathsf{ind}}, \overline{prek}_u^{\mathsf{ind}}) \overset{\$}{\leftarrow} \Sigma.\mathsf{PreKGen}()$*: outputs the* ind*-th public/private key pair of a party u.*
- ***(Ephemeral) key generation*** $(\overline{epk}_u^{\mathsf{ind}}, \overline{ek}_u^{\mathsf{ind}}) \overset{\$}{\leftarrow} \Sigma.\mathsf{EpKGen}()$*: outputs the* ind*-th public/private key pair of user u*
- ***Session execution*** $(\pi', m') \overset{\$}{\leftarrow} \Sigma.\mathsf{Run}(\overline{ik}_u, \mathcal{L}_u^{\overline{prek}}, \mathcal{L}_{\mathit{all}}^{\overline{ipk}}, \mathcal{L}_{\mathit{all}}^{\overline{prepk}}, \pi, m)$*: inputs a party u's long-term private key $\overline{ik}_u$, a list of u's private pre-keys $\mathcal{L}_u^{\overline{prek}}$, lists of long-term and medium-term public keys for all honest parties $\mathcal{L}_{\mathit{all}}^{\overline{ipk}}$ and $\mathcal{L}_{\mathit{all}}^{\overline{prepk}}$, a session state π, and an incoming message m, and outputs an updated session state π' and a (possibly empty) outgoing message m'. To set up the session sending the first message, $\Sigma.\mathsf{Run}$ is called with a distinguished message $m =$* create.
- ***Fake algorithm*** $(K, T) \overset{\$}{\leftarrow} \Sigma.\mathsf{Fake}(\overline{ipk}_u, \overline{ik}_v, \mathcal{L}_v^{\overline{prek}}, \mathsf{ind})$*: inputs one party u's long-term identity public key $\overline{ipk}_u$, the other party v's long-term identity private key $\overline{ik}_v$, a list of v's private pre-keys $\mathcal{L}_v^{\overline{prek}}$, and an index of party v's pre-key* ind *and generates a session key K and a transcript T of a protocol interaction between them.*

The session state π includes following variables (we only recall the ones related to the offline deniability):

- *role* $\in \{\mathtt{init}, \mathtt{resp}\}$: the role of the party. The initiator `init` and the responder `resp` indicate the message sender and receiver in the DAKE, respectively.
- $\mathsf{st}_{\mathsf{exec}} \in \{\bot, \mathsf{running}, \mathsf{accepted}, \mathtt{reject}\}$: The status of this session's execution. The status is initialized with $\bot$ and turns to running when the session starts. The status is set to `accept` if the DAKE is executed without errors and `reject` otherwise.

E.2. The game-based offline deniability experiment

The game-based offline deniability experiment $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$ for a DAKE protocol Σ is depicted in Figure 6, where q_{P}, q_{M}, and q_{S} respectively denotes the maximal number of parties, of (medium-term) pre-keys per party, and of total sessions. At the start of this experiment, long-term identity and medium-term pre- public/private key pairs are generated for all q_{P} honest parties and provided to the attacker[7]. A random challenge bit b is fixed for the duration of the experiment. The attacker is given repeated access to a Session-Start oracle which takes as input two party identifiers sid and rid and a pre-key index ind. If b is 0, then the Session-Start oracle will generate an honest transcript of an interaction between sid and rid using the $\Sigma.\mathsf{Run}$ algorithm and each party's secret keys. If b is 1, then the Session-Start oracle will generate a simulated transcript of an interaction between sid and rid using the $\Sigma.\mathsf{Fake}$ algorithm. At the end of the experiment, the attacker outputs a guess b′ of b. The experiment outputs 1 if b′ = b and 0 otherwise. The attacker's advantage in the deniability game is the absolute value of the difference between $\frac{1}{2}$ and the probability the experiment outputs 1.

Definition 7. *An asynchronous* DAKE *protocol Σ is $(t, \epsilon, q_{\mathsf{S}})$-deniable (with respect to maximal number of parties q_{P} and pre-keys per party q_{M}) if for any adversary $\mathcal{A}$ with running time at most t and making at most q_{S} many queries (to its* Session-Start *oracle), we have that*

$$\mathsf{Adv}^{\mathsf{deni}}_{\Sigma}(\mathcal{A}) := \left|\Pr[\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A}) = 1] - \frac{1}{2}\right| \leq \epsilon$$

where $\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})$ is defined in Figure 6.

7. The attacker here can be considered as a judge in reality.

```
$\underline{\mathsf{Exp}^{\mathsf{deni}}_{\Sigma,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}(\mathcal{A})}$:
$\mathcal{L}_{\mathsf{all}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \leftarrow \emptyset$
$\mathbf{for}\ u \in [q_{\mathsf{P}}]$
    $\mathcal{L}^{\overline{prek}}_{u} \leftarrow \emptyset$
    $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$
    $\mathcal{L}^{\overline{ipk}}_{\mathsf{all}} \xleftarrow{+} \{\overline{ipk}_u\}$
    $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{ipk}_u, \overline{ik}_u)$
    $\mathbf{for}\ \mathsf{ind} \in [q_{\mathsf{M}}]$
        $(\overline{prepk}^{\mathsf{ind}}_u, \overline{prek}^{\mathsf{ind}}_u) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$
        $\mathcal{L}^{\overline{prek}}_{u} \xleftarrow{+} \overline{prek}^{\mathsf{ind}}_u, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}} \xleftarrow{+} \overline{prepk}^{\mathsf{ind}}_u$
        $\mathcal{L}_{\mathsf{all}} \xleftarrow{+} (\overline{prepk}_u, \overline{prek}_u)$
$\mathsf{b} \xleftarrow{\$} \{0,1\}$
$\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\mathcal{L}_{\mathsf{all}})$
$\mathbf{return}\ [\![\mathsf{b} = \mathsf{b}']\!]$

$\underline{\mathsf{Session\text{-}Start}(\mathsf{sid}, \mathsf{rid}, \mathsf{ind})}$:
$\mathbf{if}\ \mathsf{b} = 0$
    $\pi_{\mathsf{rid}}.role \leftarrow \mathtt{resp}, \pi_{\mathsf{rid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $\pi_{\mathsf{sid}}.role \leftarrow \mathtt{init}, \pi_{\mathsf{sid}}.\mathsf{st}_{\mathsf{exec}} \leftarrow \mathsf{running}$
    $(\pi'_{\mathsf{rid}}, m) \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{rid}}, (\mathsf{create}, \mathsf{ind}))$
    $(\pi'_{\mathsf{sid}}, m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_{\mathsf{sid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{sid}}, \mathcal{L}^{\overline{ipk}}_{\mathsf{all}}, \mathcal{L}^{\overline{prepk}}_{\mathsf{all}}, \pi_{\mathsf{sid}}, m)$
    $(K, T) \xleftarrow{\$} (\pi'_{\mathsf{sid}}.K, (m, m'))$
$\mathbf{else}$
    $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_{\mathsf{sid}}, \overline{ik}_{\mathsf{rid}}, \mathcal{L}^{\overline{prek}}_{\mathsf{rid}}, \mathsf{ind})$
$\mathbf{return}\ (K, T)$
```

Figure 6: The offline deniability experiment for an attacker $\mathcal{A}$ against a DAKE scheme Σ. The oracle $\mathcal{O} := \{\mathsf{Session\text{-}Start}\}$.

Appendix F. Preliminaries

F.1. Key Encapsulation Mechanisms

Definition 8. *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*** $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{K.KG}(\mathsf{pp})$*: takes as input the public parameter* pp *and outputs a public encapsulation and private decapsulation key pair* $(\mathsf{ek}, \mathsf{dk})$.
- ***Encapsulation*** $(c, k) \xleftarrow{\$} \mathsf{K.Enc}(\mathsf{ek})$*: takes as input a public key* pk *and outputs a ciphertext* c *and a symmetric key* k*. We write* $(c, k) \xleftarrow{\$} \mathsf{K.Enc}(\mathsf{ek}; r^{\mathsf{Encaps}})$ *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 $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \mathsf{K.KG}()$, we have

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

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

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

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

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* (IND-CPA/IND-CCA). The IND-CPA security prevents an attacker from distinguishing the encapsulated symmetric key of a challenge ciphertext from a random one. The IND-CCA security additionally allows the attacker to access a decapsulation oracle.

Definition 9. *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-IND-XXX *secure for* $\mathsf{XXX} \in \{\mathsf{CPA}, \mathsf{CCA}\}$*, if for every (potential quantum) adversary* $\mathcal{A}$ *, we have*

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

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

F.2. Digital Signature

Definition 10. *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*** $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{D.Vrfy}(\mathsf{vk}, m, \sigma)$*: 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

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

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

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

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

Figure 7: IND-CPA and IND-CCA experiments for $\mathsf{KEM} = (\mathsf{K.KG}, \mathsf{K.Enc}, \mathsf{K.Dec})$ with symmetric key space $\mathcal{K}$.

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* EUF-CMA and SUF-CMA.

Definition 11. *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-EUF-CMA *secure (resp.* ϵ-SUF-CMA *secure), if for every (potential quantum) adversary* $\mathcal{A}$, *we have*

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

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

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 8.*

F.3. Authenticated Encryption

Definition 12. *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* $\mathsf{SKE} = (\mathsf{S.Enc}, \mathsf{S.Dec})$ *as defined below.*

- ***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 $k \xleftarrow{\$} \mathcal{K}$ and every message $m \in \mathcal{M}$, we have

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

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

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

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

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 13. *Let* $\mathsf{SKE} = (\mathsf{S.Enc}, \mathsf{S.Dec})$ *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*

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

where the $\mathrm{Expt}^{\mathsf{IND\text{-}1CCA}}_{\mathsf{SKE}}(\mathcal{A})$ *experiment is defined in Figure 9.*

F.4. Pseudorandom Generators and Pseudorandom Functions

Definition 14. *Let* $\mathsf{F} : \mathcal{R} \to \mathcal{O}$ *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*

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

Definition 15. *Let* $\mathsf{F} : \mathcal{K} \times \mathcal{M} \to \mathcal{O}$ *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* $\mathbf{R} : \mathcal{M} \to \mathcal{O}$, *we have*

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

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

Definition 16. *Let* $m \ge 2$. *Let* $\mathsf{F} : \mathcal{K}_1 \times \ldots \times \mathcal{K}_m \to \mathcal{O}$ *be a function that maps* m *keys* $k_i \in \mathcal{K}_i$ *for* $1 \le i \le m$ *to an output* $y \in \mathcal{O}$. *We say* F *is* ϵ-mprf*-secure if all of the*

```
Expt^{EUF-CMA}_{DS}(A):
    $\mathcal{L} \leftarrow \emptyset$
    $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{D.KG}()$
    $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{vk})$
    if $m^\star \in \mathcal{L}$
        return $0$
    return $[\![\mathsf{D.Vrfy}(\mathsf{vk}, m^\star, \sigma^\star)]\!]$
$\mathcal{O}_{\mathsf{Sign}}(m)$:
    $\sigma \xleftarrow{\$} \mathsf{D.Sign}(\mathsf{sk}, m)$
    $\mathcal{L} \xleftarrow{+} m$
    return $\sigma$
```

```
Expt^{SUF-CMA}_{DS}(A):
    $\mathcal{L} \leftarrow \emptyset$
    $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{D.KG}()$
    $(m^\star, \sigma^\star) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{vk})$
    if $(m^\star, \sigma^\star) \in \mathcal{L}$
        return $0$
    return $[\![\mathsf{D.Vrfy}(\mathsf{vk}, m^\star, \sigma^\star)]\!]$
$\mathcal{O}_{\mathsf{Sign}}(m)$:
    $\sigma \xleftarrow{\$} \mathsf{D.Sign}(\mathsf{sk}, m)$
    $\mathcal{L} \xleftarrow{+} (m, \sigma)$
    return $\sigma$
```

Figure 8: EUF-CMA and SUF-CMA experiments for $\mathsf{DS} = (\mathsf{D.KG}, \mathsf{D.Sign}, \mathsf{D.Vrfy})$.

```
Expt^{IND-1CCA}_{SKE}(A):
    $\mathsf{b} \xleftarrow{\$} \{0,1\}$
    $k \xleftarrow{\$} \mathcal{K}()$
    $c^\star \leftarrow \perp$
    $\mathsf{b}' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Dec}}}()$
    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 $\mathsf{S.Dec}(k, c)$
```

Figure 9: IND-1CCA experiment for $\mathsf{SKE} = (\mathsf{S.KG}, \mathsf{S.Enc}, \mathsf{S.Dec})$ with ciphertext space $\mathcal{C}$.

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 3. *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'-3prf*-secure such that* $\epsilon' \leq q\epsilon$*, where* q *denotes the number of queries by any attacker against* 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. □

Appendix G. 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 ACD19. However, we find out a minor technical flaw in their reduction, which indicates that their proposal satisfies each of their simplified security notions but *not* the original one.

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

G.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 $m' \neq \bot$ 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 $(t', i') \in \mathsf{comp}$[8], the oracle INJECT-A immediately returns $(t', i', \bot)$.

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 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 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].

8. 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.

The attacker executes as follows:

1) At epoch 0[9], 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 $c \xleftarrow{\$}$ TRANSMIT-A$(_, \bot)$ followed by DELIVER-B(c) and $c' \xleftarrow{\$}$ TRANSMIT-B$(_, \bot)$ followed by DELIVER-A(c') in turn for sufficient rounds, until the safe injection predicate safe-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, safe-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 FS-AEAD component at epoch 0 is not erased from the state, since c_1 has not been delivered[10]. 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"*
- *"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.

9. Here we use epoch 0 as example. Similar attacks also work starting at any epoch.

10. We stress that although the ACD19 includes a so-called FS-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. Notably, the same argument applies for the TR construction in [10].

Impact of our attack: Our attack effectively breaks the authenticity of not only ACD19 in [1] but also TR in [10], since the underlying FS-AEAD component of both constructions will continue to derive the message key, the index of which is even larger than the maximal length indicated by the partner.

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.

G.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 56 and 57 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 17. *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 48 is removed.
3) the attacker has to output an epoch $t^\star$ at the beginning of the experiment
4) the INJECT-A and INJECT-B 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 $t^\star$, which indicates the epoch of the forged ciphertext, without outputting another epoch $t^\star_L$ as in [1], which indicating the last corruption event before the $t^\star$.

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

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

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* $\mathsf{Exp}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

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 48 is removed.
2) the INJECT-A and INJECT-B 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, without outputting another epoch $t^\star_L$ as in [1], which indicating the last corruption event before the $t^\star$.

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 19. *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*

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

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* $\mathsf{Exp}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}}$.

Appendix H. Proof of Theorems and Lemmas

H.1. Our Lemmas

Lemma 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*

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

Lemma 2. *Let* Π *denote our* eSM *construction in Section 4.2. If the underlying* KEM*,* DS*, and* SKE *are respectively* δ_{KEM}*,* δ_{DS}*,* δ_{SKE}*-strongly correct*[11] *in time* t*, then* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \mathsf{Adv}^{\mathsf{CORR}}_{\Pi,\triangle_{\mathsf{eSM}}})$-CORR *secure for* $\triangle_{\mathsf{eSM}} = 2$*, such that*

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

Lemma 3. *Let* Π *denote our* eSM *construction in Section 4.2. If the underlying* KEM *is* $\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}$*-secure,* SKE *is* $\epsilon^{\mathsf{IND\text{-}1CCA}}_{\mathsf{SKE}}$*-secure,* KDF_1 *is* $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$*-secure*[12]*,* KDF_2 *is* $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$ *secure,* KDF_3 *is* $\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}$*-secure,* KDF_4 *is* $\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4}$*-secure,* KDF_5 *is* $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$*-secure, in time* t*, then* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}})$-PRIV *secure for* $\triangle_{\mathsf{eSM}} = 2$*, such that*

$$\mathsf{Adv}^{\mathsf{PRIV}}_{\Pi,\triangle_{\mathsf{eSM}}} \leq q_{\mathsf{M}}q_{\mathsf{ep}}q\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}} + q_{\mathsf{M}}q\epsilon^{\mathsf{IND\text{-}1CCA}}_{\mathsf{SKE}} + q_{\mathsf{M}}q_{\mathsf{ep}}q\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1} + q^2_{\mathsf{ep}}q\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2} + q_{\mathsf{ep}}q\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3} + q^2\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4} + (q_{\mathsf{M}}q_{\mathsf{ep}} + 1)q\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$$

11. By strongly correct, we mean that the schemes are conventionally correct for all randomness. See Appendix F for more details.

12. By 3prf security, we mean that a function is indistinguishable from a random function with respect to any of the three inputs. See Appendix F.4 for mode details.

Lemma 4. *Let* Π *denote our* eSM *construction in Section 4.2. If the underlying* DS *is* $\epsilon^{\mathsf{SUF\text{-}CMA}}_{\mathsf{DS}}$*-secure,* KEM *is* $\epsilon^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}}$*-secure,* SKE *is* $\epsilon^{\mathsf{IND\text{-}1CCA}}_{\mathsf{SKE}}$*-secure,* KDF_1 *is* $\epsilon^{\mathsf{3prf}}_{\mathsf{KDF}_1}$*-secure,* KDF_2 *is* $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$ *secure,* KDF_3 *is* $\epsilon^{\mathsf{prf}}_{\mathsf{KDF}_3}$*-secure,* KDF_4 *is* $\epsilon^{\mathsf{prg}}_{\mathsf{KDF}_4}$*-secure,* KDF_5 *is* $\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$*-secure, in time* t*, then* Π *is* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \mathsf{Adv}^{\mathsf{AUTH}}_{\Pi,\triangle_{\mathsf{eSM}}})$-AUTH *secure for* $\triangle_{\mathsf{eSM}} = 2$*, such that*

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

H.2. Proof of Lemma 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:

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

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 $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}$ 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, $\mathcal{B}_1$ 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(ind) and RevPreKey-B(ind): $\mathcal{B}_1$ adds the ind into the pre-key reveal list, according to the invoked oracle and runs **corruption-update**(). For each record in the 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-key counter ind 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 $t_\mathtt{P}$ into the session state corruption list $\mathcal{L}^{\mathsf{cor}}_\mathtt{P}$ and runs **corruption-update**(). Next, $\mathcal{B}_1$ 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 $\mathsf{safe}^{\mathsf{idK}}_\mathtt{P}$
  – $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{preK}}_\mathtt{P}(\mathsf{ind})$
  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 $t_\mathtt{P}$ 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(ind, m, r) and Transmit-B(ind, 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(ind, m_0, m_1, r) and Challenge-B(ind, m_0, m_1, r): We first consider the case for answering Challenge-A(ind, m_0, m_1, r). The attacker $\mathcal{B}_1$ first computes $\mathsf{flag} = [\![r = \bot]\!]$. Namely, $\mathsf{flag} = \mathsf{true}$ if and only if r is $\bot$. Then, $\mathcal{B}_1$ checks whether the predicate $\mathsf{safe\text{-}ch}_\mathtt{A}(\mathsf{flag}, t_\mathtt{A}, \mathsf{ind})$ 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(ind, m_b, r) to $\mathcal{C}_1$ for a ciphertext c. Then, $\mathcal{B}_1$ adds the record $\mathbf{record}(\mathtt{A}, \mathsf{ind}, \mathsf{flag}, t_\mathtt{A}, i_\mathtt{A}, m_\mathsf{b}, c)$ into its own allChall and chall. Finally, $\mathcal{B}_1$ returns c to $\mathcal{A}$.
  The step for answering Challenge-B(ind, m_0, m_1, r) is similar to above step except that the functions and variables related to A are replaced by the ones to 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(ind, c) and Inject-B(ind, 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 and Inject-B 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 **corruption-update**(). For each record in the 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(ind) and RevPreKey-B(ind): $\mathcal{B}_2$ adds ind into the pre-key reveal list, according to the invoked oracle and runs **corruption-update**(). For each record in the 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-key counter ind 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 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 $\mathbf{corruption\text{-}update}()$. Next, $\mathcal{B}_2$ 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, $\mathcal{B}_2$ 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(ind, m, r) and TRANSMIT-B(ind, 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$(\mathsf{ind}, m_0, m_1, r)$ and CHALLENGE-B$(\mathsf{ind}, m_0, m_1, r)$: We first consider the case for answering CHALLENGE-A$(\mathsf{ind}, m_0, m_1, r)$. The attacker $\mathcal{B}_2$ first computes $\mathsf{flag} = [\![r = \bot]\!]$. Namely, $\mathsf{flag} = \mathsf{true}$ if and only if r is $\bot$. Then, $\mathcal{B}_2$ checks whether the predicate $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$ 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$(\mathsf{ind}, m_{\mathsf{b}}, r)$ to $\mathcal{C}_2$ for a ciphertext c. Then, $\mathcal{B}_2$ adds the record $\mathbf{record}(\mathtt{A}, \mathsf{ind}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, m_{\mathsf{b}}, c)$ into its own $\mathsf{allChall}$ and chall. Finally, $\mathcal{B}_2$ returns c to $\mathcal{A}$.
The step for answering CHALLENGE-B$(\mathsf{ind}, m_0, m_1, r)$ 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(ind, c) and INJECT-B(ind, 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(ind, c) or INJECT-B(ind, 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(ind, c) and INJECT-B(ind, 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(ind, c) and INJECT-B(ind, ct), 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

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

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

Case 3. We compute the probability $|\Pr[\mathsf{Exp}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}(\mathcal{A}) = (0,0,1)] - \frac{1}{2}|$, i.e., $\mathcal{A}$ wins via the winning predicate $\mathsf{win}^{\mathsf{priv}}$ by hybrid games. Let G_j denote the simulation of **Game j**.
Game 0. This game is identical to the $\mathsf{Exp}_{\Pi,\triangle_{\mathsf{eSM}}}^{\mathsf{eSM}}$ experiment. Thus, we have that

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

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

- When the attacker queries CHALLENGE-A$(\mathsf{ind}, m_0, m_1, r)$ at epoch j, the challenger first checks whether $\mathsf{ind} \leq n_{\mathtt{B}}$ and $|m_0| = |m_1|$ 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$(\mathsf{ind}, \bar{m}, \bar{m}, r)$ 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

$$|\Pr[\mathsf{G}_{j-1}(\mathcal{A}) = (0,0,1)] - \Pr[\mathsf{G}_j(\mathcal{A}) = (0,0,1)]| \leq \Pr[E]$$

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

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

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** $(j-1)$ and **Game** j, 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 j 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** $(j-1)$, **Game** j, 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 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(ind) and REVPREKEY-B(ind): $\mathcal{B}_3$ adds ind into the pre-key reveal list, according to the invoked oracle and runs **corruption-update**(). For each record in the 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-key counter ind 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 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 $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{P}}$
  - $\mathsf{flag} = \mathsf{good}$ and $\mathsf{safe}^{\mathsf{preK}}_{\mathtt{P}}(\mathsf{ind})$

  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(ind, m, r) and TRANSMIT-B(ind, 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$(\mathsf{ind}, m_0, m_1, r)$ and CHALLENGE-B$(\mathsf{ind}, m_0, m_1, r)$: 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.
  - $[t_{\mathtt{A}} < j]$: When the attacker $\mathcal{A}$ queries CHALLENGE-A$(\mathsf{ind}, m_0, m_1, r)$ at epoch $t_{\mathtt{A}} < j$, the $\mathcal{B}_3$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next, $\mathcal{B}_3$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind}) = \mathsf{true}$, $\mathsf{ind} \leq n_{\mathtt{B}}$, and $|m_0| = |m_1|$ and aborts if any condition does not hold. Otherwise, $\mathcal{B}_3$ samples a random message $\bar{m}$ of the length $|m_0|$ and queries TRANSMIT-A$(\mathsf{ind}, \bar{m}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\mathsf{rec} = (\mathtt{A}, \mathsf{ind}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, \bar{m}, c)$ into both allChall and chall, followed by returning the ciphertext c to $\mathcal{A}$.
  - $[t_{\mathtt{A}} = j]$: When the attacker $\mathcal{A}$ queries CHALLENGE-A$(\mathsf{ind}, m_0, m_1, r)$ at epoch $t_{\mathtt{A}} = j$, the $\mathcal{B}_3$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next, $\mathcal{B}_3$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind}) = \mathsf{true}$, $\mathsf{ind} \leq n_{\mathtt{B}}$, and $|m_0| = |m_1|$ and aborts if any condition does not hold. Otherwise, $\mathcal{B}_3$ samples a random message $\bar{m}$ of the length $|m_0|$ and queries CHALLENGE-A$(\mathsf{ind}, m_{\bar{\mathsf{b}}}, \bar{m}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\mathsf{rec} = (\mathtt{A}, \mathsf{ind}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, \_, c)$ into both allChall and chall, followed by returning the ciphertext c to $\mathcal{A}$.
  - $[t_{\mathtt{A}} > j]$: When the attacker $\mathcal{A}$ queries CHALLENGE-A$(\mathsf{ind}, m_0, m_1, r)$ at epoch $t_{\mathtt{A}} > j$, the $\mathcal{B}_3$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next, $\mathcal{B}_3$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind}) = \mathsf{true}$, $\mathsf{ind} \leq n_{\mathtt{B}}$, and $|m_0| = |m_1|$, and aborts if any condition does not hold. Otherwise, $\mathcal{B}_3$ queries TRANSMIT-A$(\mathsf{ind}, m_{\bar{\mathsf{b}}}, r)$ for a ciphertext c. Finally, the $\mathcal{B}_3$ adds the record $\mathsf{rec} = (\mathtt{A}, \mathsf{ind}, \mathsf{flag}, t_{\mathtt{A}}, i_{\mathtt{A}}, m_{\bar{\mathsf{b}}}, c)$ into both 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 $(t, i, \perp)$ 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, $\mathcal{B}_3$ removes any element including (t, i, c) from the challenge set chall.
- INJECT-A(ind, c) and INJECT-B(ind, 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 and DELIVER-B and $\mathsf{win}^{\mathsf{auth}}$ in the INJECT-A and INJECT-B 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** $(j-1)$ to $\mathcal{A}$. If the challenge bit b in the PRIV experiment is 1, then $\mathcal{B}_3$ perfectly simulates **Game** j to $\mathcal{A}$. This means, the attacker $\mathcal{B}_3$ wins if and only if $\mathcal{A}$ can distinguish the adjacent hybrid games **Game** $(j-1)$ and **Game** j, 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

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

□

H.3. Proof of Lemma 2

Proof. The proof is given by a sequence of games. Let Adv_j denote the attacker $\mathcal{A}$'s advantage in winning **Game** j.
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 $t^\star \le \min(t_\mathtt{A}, t_\mathtt{B}) - 2$, the challenger immediately returns $(t^\star, i^\star, \perp)$.

Note that the oracles are defined symmetric for party A and 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 $t^\star$ 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) $\mathsf{ind} \le n_\mathtt{A}$
3) $\mathsf{safe\text{-}inj}_\mathtt{A}(t_\mathtt{B}) = \mathsf{true}$ **and** $\mathsf{safe\text{-}inj}_\mathtt{A}(t_\mathtt{A}) = \mathsf{true}$ which are equivalent to $\mathsf{safe\text{-}st}_\mathtt{B}(t_\mathtt{B}) = \mathsf{true}$ **and** $\mathsf{safe\text{-}st}_\mathtt{B}(t_\mathtt{A}) = \mathsf{true}$
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 $t^\star \le \max(t_\mathtt{A}, t_\mathtt{B})$. Moreover, a ciphertext is added into comp only when

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

In all of the above cases, we know that $\mathsf{safe\text{-}st}_\mathtt{B}(t^\star) = \mathsf{false}$. Note that the conditions $\mathsf{safe\text{-}st}_\mathtt{B}(t_\mathtt{B}) = \mathsf{false}$ **and** $\mathsf{safe\text{-}st}_\mathtt{B}(t_\mathtt{A}) = \mathsf{false}$ must hold at the same time. This means, $t^\star \le \min(t_\mathtt{A}, t_\mathtt{B}) - 2$. 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 D.Vrfy in Line 35 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 in Line 37. 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(ind, m, r) and TRANSMIT-B(ind, 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.

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

□

H.4. Proof of Lemma 3

Proof. The proof is given by a sequence of games. Let Adv_j denote the attacker $\mathcal{A}$'s advantage in winning Game j. 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$(\mathsf{ind}, m_0, m_1, r)$ 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$(\mathsf{ind}, \bar{m}, \bar{m}, r)$ instead of CHALLENGE-A$(\mathsf{ind}, m_0, m_1, r)$. 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$(\mathsf{ind}, m_0, m_1, r)$ when $i_{\mathtt{A}} < j-1$, i.e., for encrypting messages before j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$, $\mathsf{ind} \le n_{\mathtt{B}}$, and m_0 and m_1 have the same length. If any 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$(\mathsf{ind}, \bar{m}, r)$ 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$(\mathsf{ind}, m_0, m_1, r)$ when $i_{\mathtt{A}} = j-1$, i.e., for encrypting j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$, $\mathsf{ind} \le n_{\mathtt{B}}$, and m_0 and m_1 have the same length. If any 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$(\mathsf{ind}, m_{\bar{\mathsf{b}}}, \bar{m}, r)$ 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$(\mathsf{ind}, m_0, m_1, r)$ when $i_{\mathtt{A}} > j-1$, i.e., for encrypting messages after j-th message. $\mathcal{B}_1$ first computes $\mathsf{flag} \leftarrow [\![r = \perp]\!]$. Next $\mathcal{B}_1$ checks whether $\mathsf{safe\text{-}ch}_{\mathtt{A}}(\mathsf{flag}, t_{\mathtt{A}}, \mathsf{ind})$, $\mathsf{ind} \le n_{\mathtt{B}}$, and 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$(\mathsf{ind}, m_{\bar{\mathsf{b}}}, r)$ 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 $\Pr[E] \le \mathsf{Adv}_1$, and further

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

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

$\mathsf{safe\text{-}st_B}(t^\star) = \Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{idK}\Big) = \Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{preK}(\mathsf{ind}^\star)\Big) = \mathsf{false}$.

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

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

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

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

When querying identity keys or pre-keys oracles, the oracle aborts if it will triggers the safe challenge predicate $\mathsf{safe\text{-}ch_A}(\mathsf{flag}^\star, t^\star, \mathsf{ind}^\star)$ to false. When querying corruption oracles, the violation of $\mathsf{safe\text{-}st_B}$ must indicate $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{idK}\Big)$ or $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{preK}(\mathsf{ind}^\star)\Big)$. 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 $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{idK}\Big) = \mathsf{true}$ or $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{preK}(\mathsf{ind}^\star)\Big) = \mathsf{true}$ or $\mathsf{safe\text{-}st_B}(t^\star) = \mathsf{true}$ holds at the end of the experiment.

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

In this case, $\Big(\mathsf{flag}^\star = \mathsf{good}$ **and** $\mathsf{safe}_\mathsf{B}^\mathsf{idK}\Big) = \mathsf{true}$ 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 18 in Figure 4 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 37 in Figure 4 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$(\mathsf{ind}^\star, m_0, m_1, r)$ 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 18 in Figure 4 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(ind, c) oracle for a pre-key index ind and 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 25 and 49 in Figure 4 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 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 26 and 49 in Figure 4 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

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

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 $\textsc{Challenge-A}(\mathsf{ind}^\star, m_0, m_1, r)$ for encrypting $i^\star$'s message in the epoch $t^\star$, $\mathcal{B}_5$ aborts if $r \neq \perp$ 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 $\textsc{Challenge-A}$ except replacing $c' \leftarrow \mathsf{S.Enc}(mk, m)$ in Line 26 and 49 in Figure 4 by $c' \leftarrow c^\star$.
4) When $\mathcal{A}$ queries $\textsc{Deliver-B}(c)$ oracle such that c includes $t^\star$, $i^\star$, and $c^\star$, $\mathcal{B}_5$ honestly simulates $\textsc{Deliver-B}$ except for outputting $m' = \perp$.
5) When $\mathcal{A}$ queries $\textsc{Inject-B}(\mathsf{ind}, c)$ oracle for a pre-key index ind and 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 $\textsc{Inject-B}$ 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

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

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

In this case, $\left(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe}_{\mathsf{B}}^{\mathsf{preK}}(\mathsf{ind}^\star)\right) = \mathsf{true}$ holds at the end of the experiment, thus also holds at the time of challenge oracle $\textsc{Challenge-A}$ 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 $\mathtt{B}$'s identity key but $\mathtt{B}$'s $\mathsf{ind}^\star$-th pre-key. The proof can be easily given in a similar way and we have that

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

To sum up, we have that

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

Case 3: $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$**.**

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

Case 3.1: $\left(\mathsf{flag}^\star = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star)\right)$**.**

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 88 in the **ep-mgmt** helper function in Figure 4 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 21 in Figure 4 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 $\mathsf{hy}.j, (1 \leq j \leq q_{\mathsf{ep}})$: This game is identical to game **Game** $\mathsf{hy}.(j-1)$ except that:

1) When entering epoch j from $j-1$, if the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ in Line 88 in the **ep-mgmt** helper function in Figure 4 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 21 in Figure 4 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

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

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ 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 $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$ is not triggered. Note also that $\mathcal{L}_{\mathtt{P}}^{\mathsf{cor}} \xleftarrow{+} j$

is not triggered only when $\textsf{safe-ch}_{\texttt{P}}(\textsf{flag}, j-1, \textsf{ind}^\star)$, which further implies that one of the following conditions must hold: (1) $\textsf{safe-st}_{\texttt{P}}(j-1)$ or (2) $\textsf{flag} = \textsf{good}$. Then, we consider each of the two cases.

Case $\textsf{safe-st}_{\texttt{P}}(j-1)$**:** First, $\textsf{safe-st}_{\texttt{P}}(j-1)$ means $(j-1),(j-2) \notin \mathcal{L}_{\texttt{P}}^{\textsf{cor}}$. Moreover, $(j-1) \notin \mathcal{L}_{\texttt{P}}^{\textsf{cor}}$ indicates that (1) the execution $\mathcal{L}_{\texttt{P}}^{\textsf{cor}} \xleftarrow{+} (j-2)$ in **Game** $\textsf{hy}.(j-2)$ is not triggered, and (2) the state corruption on $\texttt{P}$ is not invoked during epoch $(j-1)$ and $(j-2)$. According to hybrid game **Game** $\textsf{hy}.(j-2)$, the value $\textsf{st}_{\texttt{P}}.nxs$ sampled uniformly at random during sending the first message in epoch $(j-2)$. In other words, $\textsf{st}_{\texttt{P}}.nxs$ is uniformly at random from the attacker's view when entering epoch j from $(j-1)$. During sending the first message in epoch j, $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\textsf{st}_{\texttt{P}}.nxs, r^{\textsf{KEM}}, r^{\textsf{DS}}) \leftarrow \textsf{KDF}_2(\textsf{st}_{\texttt{P}}.nxs, r)$ is executed in Line 21 in Figure 4. By the prf security of $\textsf{KDF}_2$, it is easy to know that if $\mathcal{A}$ can distinguish **Game** $\textsf{hy}.(j-1)$ and **Game** $\textsf{hy}.j$, then there must exist an attacker that distinguish the keyed $\textsf{KDF}_2$ and a random function. Thus, it holds that

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

Case $\textsf{flag} = \textsf{good}$**:** This means, the first message in epoch $j-2$ is computed using fresh randomness. In particular, this means, $r \xleftarrow{\$} \{0,1\}^\lambda$, $(\textsf{st}_{\texttt{P}}.nxs, r^{\textsf{KEM}}, r^{\textsf{DS}}) \leftarrow \textsf{KDF}_2(\textsf{st}_{\texttt{P}}.nxs, r)$ is executed in Line 21 in Figure 4 uses fresh randomness r. It is easy to know that $\textsf{st}_{\texttt{P}}.nxs$ after sending the first message in epoch $(j-2)$ is distinguishable from a random string, due to the swap-security of $\textsf{KDF}_2$.
Thus, we have that

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

From above two cases, we know that

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

To sum up, we have that

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

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

1) $\textsf{st}_{\texttt{P}}.nxs \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\textsf{KEM}} \xleftarrow{\$} \{0,1\}^\lambda$, $r^{\textsf{DS}} \xleftarrow{\$} \{0,1\}^\lambda$ are executed when $\texttt{B}$ 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 $\textsf{st}_{\texttt{B}}$ generated in epoch $t^\star - 1$ for $\texttt{A}$ 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

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

Combing the above statements, we have that

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

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

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

1) Whenever $\texttt{P} \in \{\texttt{A}, \texttt{B}\}$ is trying to sending the first message in a new epoch $t+1$ (i.e. $\texttt{P} = \texttt{A}$ if t even and $\texttt{P} = \texttt{B}$ if t odd) and the execution $\mathcal{L}_{\texttt{P}}^{\textsf{cor}} \xleftarrow{+} t+1$ in Line 88 in the **ep-mgmt** helper function in Figure 4 is not triggered, then the challenger replaces $(\textsf{st}.rk, \textsf{st}.ck^{\textsf{st}.t}) \leftarrow \textsf{KDF}_3(\textsf{st}.rk, \textsf{upd}^{\textsf{ar}})$ executed in the following eSend algorithm in Line 24 in Figure 4 by $\textsf{st}_{\texttt{P}}.rk \xleftarrow{\$} \{0,1\}^\lambda$ and $\textsf{st}.ck^{\textsf{st}.t} \xleftarrow{\$} \{0,1\}^\lambda$, followed by storing $(t+1, \textsf{st}_{\texttt{P}}.rk, \textsf{st}.ck^{t+1}, \textsf{st}.\textsf{prtr})$.
2) if there exist a locally stored tuple $(t', rk, ck, \textsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\textsf{st}.rk, \textsf{st}.ck^{\textsf{st}.t}) \leftarrow \textsf{KDF}_3(\textsf{st}.rk, \textsf{upd}^{\textsf{ar}})$ executed in the eRcv algorithm in Line 40 in Figure 4 by $\textsf{st}.rk \leftarrow rk$, $\textsf{st}.ck^{\textsf{st}.t} \leftarrow ck$.

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

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

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

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

1) When $\texttt{P} \in \{\texttt{A}, \texttt{B}\}$ is trying to send the first message in a new epoch j (i.e. $\texttt{P} = \texttt{A}$ if j odd and $\texttt{P} = \texttt{B}$ if t even) and the execution $\mathcal{L}_{\texttt{P}}^{\textsf{cor}} \xleftarrow{+} j$ in Line 88 in the **ep-mgmt** helper function in Figure 4 is not triggered, then the challenger replaces $(\textsf{st}.rk, \textsf{st}.ck^j) \leftarrow \textsf{KDF}_3(\textsf{st}.rk, \textsf{upd}^{\textsf{ar}})$ executed in the following eSend algorithm in Line 24 in Figure 4 by $\textsf{st}_{\texttt{P}}.rk \xleftarrow{\$} \{0,1\}^\lambda$ and $\textsf{st}.ck^j \xleftarrow{\$} \{0,1\}^\lambda$, followed by storing $(j, \textsf{st}_{\texttt{P}}.rk, \textsf{st}.ck^j, \textsf{st}.\textsf{prtr})$.
2) if there exist a locally stored tuple $(t', rk, ck, \textsf{prtr})$ and the eRcv is invoked to entering epoch t' with ciphertext including prtr, the challenger replaces $(\textsf{st}.rk, \textsf{st}.ck^j) \leftarrow \textsf{KDF}_3(\textsf{st}.rk, \textsf{upd}^{\textsf{ar}})$ executed in the eRcv algorithm in Line 40 in Figure 4 by $\textsf{st}.rk \leftarrow rk$, $\textsf{st}.ck^j \leftarrow ck$.

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

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

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

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

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}_{\texttt{P}}^{\textsf{cor}} \xleftarrow{+} j$ in **Game** $\textsf{hy}.j$ indicates that **Game** $\textsf{hy}.(j-1)$ is identical to **Game** $\textsf{hy}.j$. Below, we only consider the case for that the execution $\mathcal{L}_{\texttt{P}}^{\textsf{cor}} \xleftarrow{+} j$ is not triggered. Note also that $\mathcal{L}_{\texttt{P}}^{\textsf{cor}} \xleftarrow{+} j$

is not triggered only when $\mathsf{safe\text{-}ch_P}(\mathsf{flag}, j-1, \mathsf{ind})$, which further implies that one of the following conditions must hold:

1) $\left(\mathsf{safe\text{-}st_P}(j-1) \textbf{ and } \mathsf{safe\text{-}st_{\neg P}}(j-1)\right)$
2) $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st_{\neg P}}(j-1)\right)$
3) $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{idK}}_{\neg P}\right)$
4) $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{preK}}_{\neg P}(\mathsf{ind})\right)$

Then, we consider each of the four cases:

Case $\left(\mathsf{safe\text{-}st_P}(j-1) \textbf{ and } \mathsf{safe\text{-}st_{\neg P}}(j-1)\right)$**:** Recall that $\mathsf{safe\text{-}st_P}(j-1)$ and $\mathsf{safe\text{-}st_{\neg P}}(j-1)$ means $(j-1), (j-2) \notin \mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{cor}}_{\mathtt{B}}$. This indicates that (1) the execution $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}} \xleftarrow{+} (j-1)$ 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 $\mathtt{P}$ receives in the epoch $(j-1)$ is not forged by the attacker. According to hybrid game **Game** $\mathsf{hy}.(j-1)$, the value $\mathsf{st_P}.rk$ sampled uniformly at random during sending the first message in epoch $(j-1)$. In other words, $\mathsf{st_P}.rk$ 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 24 in Figure 4. 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

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

Case $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st_{\neg P}}(j-1)\right)$ **:** This case can be analyze in the following games. Here, we only sketch the idea, since they are very similar to **Game 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 $\mathsf{st}_{\neg P}$ and will be used by $\mathtt{P}$ in epoch j is sampled uniformly at random except probability $q_{\mathsf{ep}}\epsilon^{\mathsf{dual}}_{\mathsf{KDF}_2}$. Next, 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 IND-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 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 $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5} \leq \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$ due to the swap-security (and the dual-security) of the function KDF_5. Thus, we have that

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

Case $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe}^{\mathsf{idK}}_{\neg P}\right)$**:** 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 IND-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 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 $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5} \leq \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$ due to the swap-security (and the dual-security) of the function KDF_5. Thus, we have that

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

Case $\left(\mathsf{flag} = \mathsf{good} \textbf{ and } \mathsf{safe\text{-}st_{\neg P}}(j-1)\right)$ **:** 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 IND-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 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 $\epsilon^{\mathsf{swap}}_{\mathsf{KDF}_5} \leq \epsilon^{\mathsf{dual}}_{\mathsf{KDF}_5}$ due to the swap-security (and the dual-security) of the function KDF_5.

Thus, we have that

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

From above two cases, we know that

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

This means, it holds that

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

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 25 in Figure 4 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 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 47 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 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 hy.j, $(1 \leq j \leq q)$: This game is identical to game **Game** hy.$(j-1)$ except that:

1) For running 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 25 in Figure 4 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 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 47 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** 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** hy.$(j-1)$ from **Game** 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

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

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 26 and 49 in Figure 4 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 (and therefore the dual security) of underlying KDF_5. Thus, we have that

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

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 IND-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

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

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

□

H.5. Proof of Lemma 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 $t^\star \le t_{\mathtt{A}} - 2$ 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 36 in Figure 4) and will aborts the eRcv execution if i exceeds the maximal message length in the corresponding epoch (see Line 31 in Figure 4). 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) $\mathsf{ind} \le n_{\mathtt{A}}$
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) $m' \ne \perp$
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) $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{true}$ holds at the time of $\mathtt{B}$ sending message corresponding to the position $(t^\star, i^\star)$, and
2) if $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{false}$, $\textsc{Corrupt-A}$ cannot be queried
3) If $\textsc{Corrupt-A}$ is queried at epoch $t^\star$, then $\textsc{Corrupt-B}$ cannot be queried.
4) $\textsc{Corrupt-B}$ 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, $\mathsf{safe\text{-}st}_{\mathtt{B}}(t^\star) = \mathsf{false}$

So, at most one of $\textsc{Corrupt-A}$ and $\textsc{Corrupt-B}$ at epoch $t^\star$, but not both.

We separate the analysis for $t^\star \ge t_{\mathtt{A}} - 1$, see Case 1, or $t^\star \le t_{\mathtt{A}} - 2$, see Case 2.

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

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

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 $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}} \xleftarrow{+} t + 1$ in Line 88 in the **ep-mgmt** helper function in Figure 4 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 21 in Figure 4 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 C1.2** by hybrid games.

Game hy.0: This game is identical to **Game 1**. Thus, we have that

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

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

1) When entering epoch j from $j - 1$, if the execution $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}} \xleftarrow{+} j$ in Line 88 in the **ep-mgmt** helper function in Figure 4 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 21 in Figure 4 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** hy.q_{ep} is identical to **Game C1.2**. Thus, we have that

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

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

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

Below, we compute the probability of the occurrence of event E by case distinction. Note that the execution $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}} \xleftarrow{+} j$ in **Game** hy.j indicates that **Game** hy.$(j-1)$ is identical to **Game** 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 $\mathsf{safe\text{-}ch_P}(\mathsf{flag}, j-1, \mathsf{ind})$ for some pre-key index ind, 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 $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} (j-2)$ in **Game** 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** 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, $r \xleftarrow{\$} \{0,1\}^{\lambda}$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 21 in Figure 4. By the prf security of KDF_2, it is easy to know that if $\mathcal{A}$ can distinguish **Game** hy.$(j-1)$ and **Game** 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, $r \xleftarrow{\$} \{0,1\}^{\lambda}$, $(\mathsf{st_P}.nxs, r^{\mathsf{KEM}}, r^{\mathsf{DS}}) \leftarrow \mathsf{KDF}_2(\mathsf{st_P}.nxs, r)$ is executed in Line 21 in Figure 4 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 $t_{\mathtt{A}} - 1 \leq t^{\star}$ and that $t^{\star}$ even. Then, there are following seven cases:

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

In all of above seven cases, $t^{\star}$ and $t_{\mathtt{B}}$ are not two epochs apart. Moreover, by $\mathsf{safe\text{-}st_B}(t_{\mathtt{A}})$ amd $\mathsf{safe\text{-}st_B}(t_{\mathtt{B}})$, we know that the $\mathcal{A}$ has to forge at least one 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 SUF-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 $\mathsf{safe\text{-}st_B}(t^{\star}) = \mathsf{true}$, where $(t^{\star}, i)$ is the position of c.

Note that $\mathsf{safe\text{-}st_B}(t^{\star}) = \mathsf{true}$ holds at the time of $\mathtt{B}$ sending message corresponding to the position $(t^{\star}, i^{\star})$ for some $i^{\star}$. This means, $\mathsf{safe\text{-}st_B}(t^{\star}) = \mathsf{true}$ 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 $\mathsf{safe\text{-}st_B}(t^{\star}) = \mathsf{true}$ 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 SUF-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 $\mathsf{safe\text{-}st_B}(t^{\star}) = \mathsf{false}$ 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 $\mathsf{st_B}$. 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 $\mathcal{L}_{\mathsf{P}}^{\mathsf{cor}} \xleftarrow{+} t+1$ in Line 88 in the **ep-mgmt** helper function in Figure 4 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 24 in Figure 4 by $\mathsf{st_P}.rk \xleftarrow{\$} \{0,1\}^{\lambda}$ and $\mathsf{st}.ck^{\mathsf{st}.t} \xleftarrow{\$} \{0,1\}^{\lambda}$, followed by storing $(t+1, \mathsf{st_P}.rk, \mathsf{st}.ck^{t+1}, \mathsf{st.prtr})$.
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 40 in Figure 4 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 H.4. We can easily know that

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

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

1) For running 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 25 in Figure 4 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 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 47 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 26 and 49 in Figure 4 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 (and therefore the dual security) of underlying KDF_5. Thus, we have that

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

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 IND-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 IND-1CCA security of SKE. The reduction simulates **Game C2.7** honestly except for the TRANSMIT-B(ind, m, r) and INJECT-A(ind, c) that is described above.

For the TRANSMIT-B(ind, 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

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

The following equation concludes the proof.

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

□

H.6. Proof of Theorem 2

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 4.2, then we can always construct an attacker $\mathcal{B}$ that breaks the offline deniability of Σ in terms of Definition 7, also see [11, 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 $\Big((\mathsf{st}_{\mathtt{A}}, \mathsf{id}_{\mathtt{A}}), (\mathsf{st}_{\mathtt{B}}, \mathsf{id}_{\mathtt{B}})\Big) \leftarrow \mathsf{st}_{\mathsf{Fake}}$
2) **if** $\mathsf{id}_{\mathtt{A}} = \mathsf{sid}$, **then**
   a) $(\mathsf{st}_{\mathtt{A}}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}_{\mathtt{A}}, ipk, prepk, m)$
   b) copy all symmetric values in session state $\mathsf{st}_{\mathtt{A}}$ to session state $\mathsf{st}_{\mathtt{B}}$
   c) If $\mathsf{st}_{\mathtt{A}}.t$ is incremented in the above $\Pi.\mathsf{eSend}$ invocation, then extract the new verification key vk and new encryption key ek from $c$, followed by set vk and ek into $\mathsf{st}_{\mathtt{B}}$
   d) $\mathsf{st}_{\mathsf{Fake}} \leftarrow ((\mathsf{st}_{\mathtt{A}}, \mathsf{id}_{\mathtt{A}}), (\mathsf{st}_{\mathtt{B}}, \mathsf{id}_{\mathtt{B}}))$
3) **else**
   a) $(\mathsf{st}_{\mathtt{B}}, c) \xleftarrow{\$} \Pi.\mathsf{eSend}(\mathsf{st}_{\mathtt{B}}, ipk, prepk, m)$
   b) copy all symmetric values in session state $\mathsf{st}_{\mathtt{B}}$ to session state $\mathsf{st}_{\mathtt{A}}$
   c) If $\mathsf{st}_{\mathtt{B}}.t$ is incremented in the above $\Pi.\mathsf{eSend}$ invocation, then extract the new verification key vk and new encryption key ek from $c$, followed by set vk and ek into $\mathsf{st}_{\mathtt{A}}$
   d) $\mathsf{st}_{\mathsf{Fake}} \leftarrow ((\mathsf{st}_{\mathtt{A}}, \mathsf{id}_{\mathtt{A}}), (\mathsf{st}_{\mathtt{B}}, \mathsf{id}_{\mathtt{B}}))$
```

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 $\mathsf{Exp}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}^{\mathsf{deni}}$. 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 Session-Start(sid, rid, aid, did, 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 (sid, rid, ind) for the key K and the transcript T. After that, $\mathcal{B}$ runs the above defined function $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}_{\mathsf{aid}}^{prek}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ for a fake state $\mathsf{st}_{\mathsf{Fake}}^{n}$. Finally, $\mathcal{B}$ returns the transcript to $\mathcal{A}$.

When $\mathcal{A}$ queries Session-Execute(sid, rid, i, ind, m), $\mathcal{B}$ simply simulates Session-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 $\mathsf{Fake}_{\Pi}^{\mathsf{eInit}}$ algorithm perfectly simulates the process of running Π.eInit-A and Π.eInit-B. Moreover, we consider two cases for the queries to the Session-Execute oracle:

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

To sum up, in both cases $\mathcal{B}$ perfectly simulates $\mathsf{Exp}_{\Sigma,\Pi,q_{\mathsf{P}},q_{\mathsf{M}},q_{\mathsf{S}}}^{\mathsf{deni}}$ 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. $\square$

Contents