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

Cas Cremers
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
cremers@cispa.de

Mang Zhao
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
mang.zhao@cispa.de

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 adversary 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 adversaries 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.

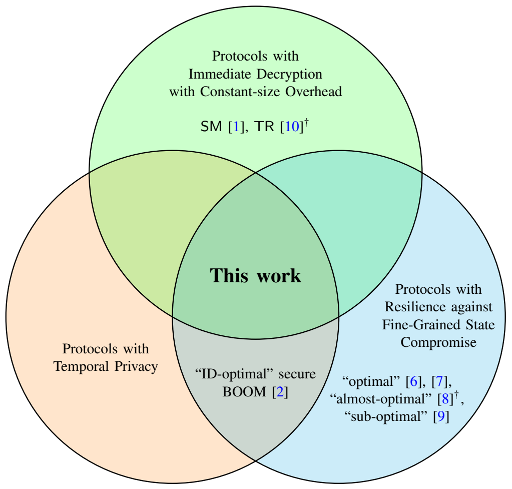

Figure 1: Comparison between this work and other existing protocols with provable security properties w.r.t. (i) immediate decryption with constant-size overhead, (ii) temporal privacy, and (iii) resilience against fine-grained state compromise. All constructions in the diagram (including this work) 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 and provide the full proofs of our lemmas and theorems in [12].

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], [13], and *offline deniability* [14]. 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 in our full version [12].

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 adversary 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], [15], [16]. 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 provide 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 adversary 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 adversary 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 [17] and [14] 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 [18]–[21] 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. [22] 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) adversaries.

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* eSM = (IdKGen, PreKGen, eInit-A, eInit-B, eSend, eRcv)*, where*

- $(ipk, ik) \overset{\$}{\leftarrow} \mathsf{IdKGen}()$ *outputs an long-term identity public-private key pair,*
- $(prepk, prek) \overset{\$}{\leftarrow} \mathsf{PreKGen}()$ *outputs a medium-term public-private pre-key pair,*
- $\mathsf{st}_\mathtt{A} \leftarrow \mathsf{eInit\text{-}A}(iss)$ *(resp.* $\mathsf{st}_\mathtt{B} \leftarrow \mathsf{eInit\text{-}B}(iss)$*) inputs an initial shared secret iss* $\in \mathcal{ISS}$ *and outputs a session state,*
- $(\mathsf{st}', c) \overset{\$}{\leftarrow} \mathsf{eSend}(\mathsf{st}, ipk, prepk, m)$ *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 A and 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 B acts as the sender and 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.

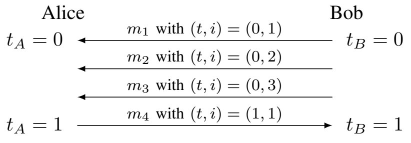
Alice
Bob
$t_A = 0$
$t_B = 0$
m_1 with $(t, i) = (0, 1)$
m_2 with $(t, i) = (0, 2)$
m_3 with $(t, i) = (0, 3)$
m_4 with $(t, i) = (1, 1)$
$t_A = 1$
$t_B = 1$

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 adversary 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 adversary.
4) **Post-compromise security** (PCS)**:** The parties can recover from session state compromise (assuming the access to fresh randomness) when the adversary is passive.

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

5) ***Strong* authenticity:** The adversary 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 adversary 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 adversary 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 adversary 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 B initializes a conversation with A using X3DH in Signal. The leakage of the sender 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 B continuously sends messages to A without receiving a reply in Signal, all messages in the sequence are leaked, since the adversary can use 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 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.

Remark 1. *The relation between* PPR *and* PCS *depends on what we take as the reference point for* PCS. *The term "Post Compromise Security" was introduced in 2016 in [23], which defines both a broader informal security guarantee as well as a specific instantiation.* PPR *can be seen as a subclass of the general initial* PCS *notion from [23].*

Over time, follow-up works have developed more fine-grained notions of PCS, *notably instantiated for specific protocol classes. One such example is [1], whose target protocol class closely matches ours. Compared to the* PCS *instantiation in [1],* PPR *can be regarded as an orthogonal class of privacy that is related to time (aka. temporal privacy). Although both the* PCS *instance from [1] and* PPR *provide healing after compromise and might look similar, they differ in the following three aspects.*

1) ***Different Healing Objects****: While the* PCS *instance from [1] heals the session state (e.g., encryption/decryption keys), and might further impact on other security guarantees (e.g. privacy, authenticity, etc.),* PPR *heals the (strong) privacy, which is a concrete security guarantee.*
2) ***Different Healing Approaches****: The* PCS *instance from [1] holds only when the session states are healed. Note that (strong) privacy is expected to hold "unless the receiver's session state, private identity key, and corresponding private pre-key all are compromised". Thus,* PPR *might hold when some private materials other than session states are recovered, i.e., is independent of their instance of* PCS.

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* $m = \perp$,
2) *the values* (t, i) *output by* eRcv *can be efficiently computed from* c,
3) *if* eRcv *has already accepted an ciphertext corresponding to the position* (t, i), *the next ciphertext corresponding to the same position 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.
- $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}, \mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}$: 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.

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.

- 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 adversary 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 adversary 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.

- $\mathbf{sam\text{-}if\text{-}nec}(r)$: If $r \neq \perp$, this function outputs (r, bad) indicating that the randomness is adversary-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 $\mathsf{type} = \mathsf{chall}$, 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 $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}}$ 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.

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.

- $\mathbf{delete}(t, i)$: 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 $\mathbf{corruption\text{-}update}$ 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 $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ 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 adversary is given access to $\mathcal{O}_1 := \{\text{NewIdKey-A}, \text{NewIdKey-B}, \text{NewPreKey-A}, \text{NewPreKey-B}\}$ 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 adversary 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 $\mathtt{A}$. The oracles for party $\mathtt{B}$ 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.

- NewIdKey-A(r), 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.
- NewPreKey-A(r), NewPreKey-B(r): Similar to the oracles above, a public-private pre-key pair is generated. The corresponding pre-key index is added into the list $\mathcal{L}^{\mathsf{rev}}_{\mathtt{A}}$ or $\mathcal{L}^{\mathsf{rev}}_{\mathtt{B}}$ if the input $r \neq \perp$. The public key is returned.
- RevIdKey-A, RevIdKey-B: These oracles simulate the reveal of the identity private key of a party $\mathtt{P} \in \{\mathtt{A}, \mathtt{B}\}$. The corresponding safe predicate is set to false. Then, the $\mathbf{corruption\text{-}update}$ 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 adversary 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 adversary from making a trivial forgery by using the information

leaked by the reveal of the identity key.
Finally, the corresponding private identity key is returned.

- RevPreKey-A(n), 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 $\mathcal{L}^{\mathsf{rev}}_{\mathtt{P}}$. 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 $\mathsf{allChall}$ 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 adversaries to corrupt session states.

- Corrupt-A, 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 $\mathcal{L}^{\mathsf{cor}}_{\mathtt{P}}$, 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 adversary from distinguishing the challenge bit by trivially revealing enough information to decrypt the past challenge ciphertexts.
After that, we add all records $\mathsf{rec} \in \mathsf{trans}$, which are produced by $\neg\mathtt{P}$ at an unsafe epoch t, into the compromise set comp. We also add all records $\mathsf{rec} \in \mathsf{trans}$, which are produced by $\mathtt{P}$ at current epoch if the partner's session at current epoch is not safe. This requirement prevents the adversary 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 adversary's capability of manipulating the message transmission.

- Transmit-A(ind, m, r), Transmit-B(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.
- Challenge-A$(\mathsf{ind}, m_0, m_1, r)$, Challenge-B$(\mathsf{ind}, m_0, m_1, r)$: These challenge oracles simulate the sending execution, where the adversary 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 $\mathsf{safe\text{-}ch}_{\mathtt{P}}(\mathsf{flag}, t_{\mathtt{P}}, \mathsf{ind})$ 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.
- Deliver-A(ind, c), Deliver-B(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 adversary 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 adversary 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.
- Inject-A(ind, c), Inject-B(ind, c): These oracles simulate a party $\mathtt{P}$'s receiving execution of a ciphertext forged by the adversary. 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 $\mathsf{safe\text{-}inj}_{\mathtt{P}}(t_{\mathtt{A}})$ and $\mathsf{safe\text{-}inj}_{\mathtt{P}}(t_{\mathtt{B}})$ both are true. If the decrypted message is not $\perp$ and the ciphertext at the same position is not compromised, the adversary 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* $(t, q, q_{\mathsf{ep}}, q_{\mathsf{M}}, \triangle_{\mathsf{eSM}}, \epsilon)$-$\mathsf{eSM}$ *secure if the below defined advantage for all adversaries against the* $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$ *experiment in Figure 3 in time* t *is bounded by*

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

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 adversary invokes the transmission and delivery oracles in an arbitrary order, which causes the winning event via Line 48.
- **Forward secrecy**: Note that the adversary can freely access the corruption oracles if all challenge ciphertexts have been delivered. No FS means that the adversary can distinguish

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 -$\mathcal{A}$). 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.

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 adversary 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 adversary 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 adversary via Line 57.
- **Strong privacy**: Note that the challenge ciphertexts must be produced without the violation the safety predicate $\mathsf{safe\text{-}ch}$ 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 adversary 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 adversary 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 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 adversary. This means that the adversary can only query $\textsc{NewIdKey-A}(r)$ with input $r = \perp$ and never query $\textsc{RevIdKey-A}$ oracle in $\mathsf{Exp}^{\mathsf{eSM}}_{\Pi,\triangle_{\mathsf{eSM}}}$. Thus, the predicate $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{A}}$ is always true. If the partner $\mathtt{B}$ 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 [24], 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.

We give a detailed comparison with the ACD19 construction [1] in our full version [12, Appendix D].

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* $\mathsf{st_A.id} = \mathtt{A}$ *and* $\mathsf{st_B.id} = \mathtt{B}$.
- $\mathsf{st}.t$*: the local epoch counter. It starts with* 0.
- $\mathsf{st}.i^0$, $\mathsf{st}.i^1$, ...*: 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 (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.

- $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$: This algorithm is called in eRcv algorithm when the caller switches its role from message sender in epoch $\mathsf{st}.t$ to 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., $\mathsf{st}.i^{t'}$, $\mathsf{st}.ck^{t'}$, $\mathsf{st.dk}^{t'}$, $\mathsf{st.vk}^{t'}$, $\mathsf{st}.\mathcal{D}^{t'}_{urk}$, $\mathsf{st}.\mathcal{D}_l[t']$ are erased, i.e., set to $\bot$. Moreover, the number how many times the chain key $\mathsf{st}.ck^{\mathsf{st}.t}$ has been

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

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

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

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

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

eRcv(st, ik, prek, c):
  $((\mathsf{prtr^{ar}}, \sigma^{\mathsf{ar}}), \mathsf{prtr^{ur}}, \sigma^{\mathsf{ur}}) \leftarrow c$
  $(l, c_1, c_2, c_3, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{prtr^{ar}}$, $(t, i, c', c'_1, c'_2, c'_3) \leftarrow \mathsf{prtr^{ur}}$
  $\mathbf{if}\ t \leq \mathsf{st}.t - 2$: $\mathbf{req}\ \mathsf{st}.\mathcal{D}_l[t] \neq \bot\ \mathbf{and}\ i \leq \mathsf{st}.\mathcal{D}_l[t]$
  $\mathbf{req}\ t \leq \mathsf{st}.t + 1$
  $\mathbf{req}\ (\mathsf{st.id} = \mathtt{A}\ \mathbf{and}\ t\ \ even)\ \mathbf{or}\ (\mathsf{st.id} = \mathtt{B}\ \mathbf{and}\ t\ \ odd)$
  $\mathbf{if}\ t = \mathsf{st}.t + 1$
    $\mathbf{req}\ \mathsf{D.Vrfy}(\mathsf{st.vk}^{t-2}, \mathsf{prtr^{ar}}, \sigma^{\mathsf{ar}})$
    $\mathsf{eRcv\text{-}Max}(\mathsf{st}, l)$, $\mathsf{st}.\mathcal{D}_l[t-2] \leftarrow l$, $\mathsf{st}.t{+}{+}$
    $k_1 \leftarrow \mathsf{K.Dec}(\mathsf{st.dk}^{\mathsf{st}.t}, c_1)$, $k_2 \leftarrow \mathsf{K.Dec}(ik, c_2)$
    $k_3 \leftarrow \mathsf{K.Dec}(prek, c_3)$
    $(\mathsf{upd^{ar}}, \_) \leftarrow \mathsf{KDF}_1(k_1, k_2, k_3)$
    $(\mathsf{st}.rk, \mathsf{st}.ck^{\mathsf{st}.t}) \leftarrow \mathsf{KDF}_3(\mathsf{st}.rk, \mathsf{upd^{ar}})$
    $\mathcal{D}^{\mathsf{st}.t}_{urk}[\cdot] \leftarrow \bot$, $\mathsf{st}.i^{\mathsf{st}.t} \leftarrow 0$, $\mathsf{st.ek}^{\mathsf{st}.t+1} \leftarrow \mathsf{ek}$, $\mathsf{st.vk}^{\mathsf{st}.t} \leftarrow \mathsf{vk}$
  $\mathbf{req}\ \mathsf{D.Vrfy}(\mathsf{st.vk}^{t}, \mathsf{prtr^{ur}}, \sigma^{\mathsf{ur}})$
  $k'_1 \leftarrow \mathsf{K.Dec}(\mathsf{st.dk}^{t}, c'_1)$, $k'_2 \leftarrow \mathsf{K.Dec}(ik, c'_2)$
  $k'_3 \leftarrow \mathsf{K.Dec}(prek, c'_3)$
  $(\_, \mathsf{upd^{ur}}) \leftarrow \mathsf{KDF}_1(k'_1, k'_2, k'_3)$
  $\mathbf{while}\ \mathsf{st}.i^t \leq i$
    $(\mathsf{st}.ck^t, urk) \leftarrow \mathsf{KDF}_4(\mathsf{st}.ck^t)$, $\mathcal{D}^t_{urk}[\mathsf{st}.i^t] \leftarrow urk$, $\mathsf{st}.i^t{+}{+}$
  $urk \leftarrow \mathcal{D}^t_{urk}[i]$, $\mathcal{D}^t_{urk}[i] \leftarrow \bot$, $\mathbf{req}\ urk \neq \bot$
  $mk \leftarrow \mathsf{KDF}_5(urk, \mathsf{upd^{ur}})$ , $m \leftarrow \mathsf{S.Dec}(mk, c')$
  $\mathbf{return}\ (\mathsf{st}, t, i, m)$
```

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.

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.

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 $\mathsf{st}.\mathsf{ek}^{\mathsf{st}.t}$ is erased.

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

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

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

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

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

Finally, $\mathtt{A}$ sets the identity $\mathsf{st}_\mathtt{A}.\mathsf{id}$ to $\mathtt{A}$, the local pre-transcript $\mathsf{st}_\mathtt{A}.\mathsf{prtr}$ to $\perp$, the epoch counter $\mathsf{st}_\mathtt{A}.t$ to 0, the message index $\mathsf{st}_\mathtt{A}.i^0$ 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 $\mathsf{st}_\mathtt{A}$.

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

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

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

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

If the caller switches its role from receiver to sender, i.e. the caller $\mathsf{st}.\mathsf{id}$ is $\mathtt{A}$ and the epoch $\mathsf{st}_\mathtt{A}.t$ is even or the caller is $\mathtt{B}$ and the epoch is odd, it first executes the following so-called *asymmetric ratchet* (ar) framework: First, the caller runs $\mathsf{eSend\text{-}Stop}(\mathsf{st})$ for a value l that counts the sent messages in the previous epoch, followed by incrementing the epoch counter $\mathsf{st}.t$ by 1 and initializing the message index counter $\mathsf{st}.i^{\mathsf{st}.t}$ 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 $\mathsf{st}.\mathsf{sk}^{\mathsf{st}.t-2}$ for a signature σ^{ar}. The pre-transcript $\mathsf{prtr}^{\mathsf{ar}}$ and signature σ^{ar} are stored into the state $\mathsf{st}.\mathsf{prtr}$. 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 $\mathsf{st}.\mathsf{sk}^{\mathsf{st}.t}$. This algorithm outputs a new state st and a final ciphertext, which is a tuple of the ar pre-transcript and signature $\mathsf{st}.\mathsf{prtr} = (\mathsf{prtr}^{\mathsf{ar}}, \sigma^{\mathsf{ar}})$, 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 $\mathtt{B}$ if the parsed epoch counter is even or odd) and in a meaningful range (by checking whether $t \leq \mathsf{st}.t+1$). If any check is wrong, the eRcv aborts and outputs $m = \perp$.

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 $\mathcal{D}_{urk}^{\mathsf{st}.t}$ 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 $\epsilon_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}$-secure,* DS *is δ_{DS}-strongly correct and $\epsilon_{\mathsf{DS}}^{\mathsf{SUF\text{-}CMA}}$-secure,* SKE *is δ_{SKE}-strongly correct and $\epsilon_{\mathsf{SKE}}^{\mathsf{IND\text{-}1CCA}}$-secure,* KDF_1 *is $\epsilon_{\mathsf{KDF}_1}^{\mathsf{3prf}}$-secure*[5], KDF_2 *is $\epsilon_{\mathsf{KDF}_2}^{\mathsf{dual}}$ secure,* KDF_3 *is $\epsilon_{\mathsf{KDF}_3}^{\mathsf{prf}}$-secure,* KDF_4 *is $\epsilon_{\mathsf{KDF}_4}^{\mathsf{prg}}$-secure,* KDF_5 *is $\epsilon_{\mathsf{KDF}_5}^{\mathsf{dual}}$-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*

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

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 in our full version [12].

Second, we introduce four lemmas in Appendix H.1 in our full version [12]. Lemma 1 reduces the eSM security to the simplified security notions, the full proof of which is given in Appendix H.2 in our full version [12]. 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 in our full version [12]. 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 [25], and the NIST suggested CRYSTALS-Dilithium for the PQ security, which is analyzed in [26]. A generic approach to instantiating KEM is to encrypt random strings using deterministic OW-CCA or merely OW-CPA secure PKE for strong correctness [27], [28]. The NIST suggested NTRU is also available for IND-CCA security and strong correctness [29]. The deterministic IND-1CCA secure authenticated encryption SKE can be instantiated with the Encrypt-then-MAC construction in [30]. 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 in our full version [12]. We suggest to double the security parameter of the symmetric primitives for PQ security.

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

4. By strongly correct, we mean that the schemes are conventionally correct for all randomness. See Appendix F in [12].

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 in [12].

the original model in [11, Definition 11], also see Definition 7 in Appendix D, is highlighted with blue color.

In addition to message senders and receivers, the deniability experiment includes three new roles: accuser, defendant, and judge. For any two-party conversation, we call a party "accuser", whose identifier is denoted by aid, if it wants to accuse that its honest conversation partner has communicated with it. Correspondingly, we call the accused honest partner "defendant", whose identifier is denoted by did. The role of the "judge" in the experiment is performed by the adversary. The goal of the experiment is to ensure that no adversaries (i.e., the real-life judges) can distinguish the real conversation transcripts between accusers and defendants from the fake ones that are produced by the accusers alone, given all secrets of all parties.

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 adversary (e.g., the judge). A challenge bit $\mathsf{b} \in \{0,1\}$ is randomly sampled. The adversary (i.e., the judge) is given repeated access to the following two oracles: The Session-Start oracle initializes a session between a sender sid and a receiver rid and determines that the party $\mathsf{aid} \in \{\mathsf{sid}, \mathsf{rid}\}$ plays the role of accuser in this session and the other party $\mathsf{did} \in \{\mathsf{sid}, \mathsf{rid}\}$ plays the role of defendant in this session. This oracle executes a real session setup and real eSM initialization if $\mathsf{b} = 0$, and some fake algorithms that simulate the accuser's view if $\mathsf{b} = 1$. The Session-Execute forwards the interaction in an existing session one step: this oracle executes eSM algorithm for sending and receiving one message honestly if $\mathsf{b} = 0$, and some fake algorithms that simulate the accuser's view if $\mathsf{b} = 1$. The adversary wins if it can distinguish real conversation transcripts (i.e., $\mathsf{b} = 0$) from fake transcripts that simulate accusers' views (i.e., $\mathsf{b} = 1$). We say a full messaging protocol is offline deniable, if there exist fake algorithms that prevent all adversaries from winning the offline deniability experiment in polynomial time. By this, we ensure that if a protocol is offline deniable, then no judge can 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 $\mathsf{Fake}^{\mathsf{eInit}}_{\Pi}(K, ipk_{\mathsf{did}}, ik_{\mathsf{aid}}, \mathcal{L}^{prek}_{\mathsf{aid}}, \mathsf{sid}, \mathsf{rid}, \mathsf{aid}, \mathsf{did})$ to produce a fake state $\mathsf{st}^n_{\mathsf{Fake}}$. The transcript T is returned.

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 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 adversary (e.g. the judge) to obtain *all* initial private secret of all parties, as in [11].

Second, while the model in [11] prevents an adversary from deciding the challenge bit b given the (output) shared keys and the transcripts of DAKE key establishments, our model prevents an adversary 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 [14].

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 $\Sigma.\mathsf{Fake}$ 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].

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 adversary* $\mathcal{A}$ *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.*

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.

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

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

Figure 5: The offline deniability experiment for an adversary $\mathcal{A}$ 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.

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 adversary 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 in our full version [12]. □

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] C. Cremers and M. Zhao, "Secure messaging with strong compromise resilience, temporal privacy, and immediate decryption (full version)," Cryptology ePrint Archive, Paper 2022/1481, 2022, https://eprint.iacr.org/2022/1481. [Online]. Available: https://eprint.iacr.org/2022/1481

[13] 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

[14] 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.

[15] 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.

[16] 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.

[17] 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

[18] 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

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

[20] 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.

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

[22] 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.

[23] K. Cohn-Gordon, C. Cremers, and L. Garratt, "Post-Compromise Security," Cryptology ePrint Archive, Paper 2016/221, 2016, https://eprint.iacr.org/2016/221. [Online]. Available: https://eprint.iacr.org/2016/221

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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

[29] 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.

[30] 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.

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 in our full version [12].

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 adversary can intercept a message and corrupt the receiver's state in the future, the adversary 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 $\mathsf{safe}^{\mathsf{idK}}_{\mathtt{P}}$, $\mathcal{L}^{\mathsf{rev}}_{\mathtt{P}}$, 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 $\mathcal{L}^{\mathsf{cor}}_{\mathtt{A}}$ and $\mathcal{L}^{\mathsf{cor}}_{\mathtt{B}}$ 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 $\mathsf{safe}^{\mathsf{preK}}_{\mathtt{P}}$ 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 $\mathsf{rec} \in \mathsf{trans}$, which are produced by $\neg\mathtt{P}$ at an unsafe epoch t (but not all epochs as in [1]), into the compromise set comp.

Compared to [1], the corruption oracles in our model can be queried under weaker requirements, providing the adversary with more information. Moreover, our corruption oracles set fewer records into the compromise set, which enables the adversary 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. Review on DAKE Scheme and the Game-based Deniability

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

D.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*** $(\overline{ipk}_u, \overline{ik}_u) \xleftarrow{\$} \Sigma.\mathsf{IdKGen}()$*: outputs the identity public/private key pair of a party* u*.*
- ***(Medium-term) pre-key generation*** $(\overline{prepk}^{\mathsf{ind}}_u, \overline{prek}^{\mathsf{ind}}_u) \xleftarrow{\$} \Sigma.\mathsf{PreKGen}()$*: outputs the* ind*-th public/private key pair of a party* u*.*
- ***(Ephemeral) key generation*** $(\overline{epk}^{\mathsf{ind}}_u, \overline{ek}^{\mathsf{ind}}_u) \xleftarrow{\$} \Sigma.\mathsf{EpKGen}()$*: outputs the* ind*-th public/private key pair of user* u
- ***Session execution*** $(\pi', m') \xleftarrow{\$} \Sigma.\mathsf{Run}(\overline{ik}_u, \mathcal{L}^{\overline{prek}}_u, \mathcal{L}^{\overline{ipk}}_{\mathit{all}}, \mathcal{L}^{\overline{prepk}}_{\mathit{all}}, \pi, m)$*: inputs a party* u*'s long-term private key*

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

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

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

$\overline{ik}_u$, a list of u's private pre-keys $\mathcal{L}^{\overline{prek}}_u$, lists of long-term and medium-term public keys for all honest parties $\mathcal{L}^{\overline{ipk}}_{\mathsf{all}}$ and $\mathcal{L}^{\overline{prepk}}_{\mathsf{all}}$, 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, Σ.Run is called with a distinguished message $m =$ create.

- ***Fake algorithm** $(K, T) \xleftarrow{\$} \Sigma.\mathsf{Fake}(\overline{ipk}_u, \overline{ik}_v, \mathcal{L}^{\overline{prek}}_v, \mathsf{ind})$: 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 $\mathcal{L}^{\overline{prek}}_v$, 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.
- $\mathsf{st}_{\mathsf{exec}} \in \{\perp, \mathsf{running}, \mathsf{accepted}, \mathtt{reject}\}$: The status of this session's execution. The status is initialized with $\perp$ and turns to running when the session starts. The status is set to `accept` if the DAKE is executed without errors and `reject` otherwise.

D.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 adversary[7]. A random challenge bit b is fixed for the duration of the experiment. The adversary 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 Σ.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 Σ.Fake algorithm. At the end of the experiment, the adversary outputs a guess b$'$ of b. The experiment outputs 1 if b$'$ = b and 0 otherwise. The adversary'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 adversary here can be considered as a judge in reality.

Appendix E. Meta-Review

E.1. Summary

This paper introduces a new secure messaging protocol (eSM) that simultaneously achieves strong compromise resilience with temporal privacy and immediate decryption with constant size overhead. The paper introduces a new model called extended-SM to prove their construction achieves these notions, and a offline deniable (OD) framework model to analyse the deniability of a composition of eSM with a deniable authenticated key exchange (DAKE) protocol.

E.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field
- Establishes a New Research Direction

E.3. Reasons for Acceptance

1) The paper extends state-of-the-art by achieving stronger notions of compromise resilience than existing (practical) solutions, but unlike previous optimal SM protocols, is able to achieve this with constant size overhead. eSM is able to provably achieve these desirable properties simultaneously for the first time, in addition to upgrading to PQ security straightforwardly.
2) The paper introduces an extended-SM model that captures these notions simultaneously, is modular enough to be applied to different SM protocols, and the authors highlight the specific additions to the model compared to the original SM model.
3) The paper explores the offline deniability of eSM in practice by using an OD framework to analyse the composition of eSM with a DAKE protocol, and proves that the composition achieves OD.

E.4. Noteworthy Concerns

The authors seek to solve a practical problem (exploring the tradeoff between strong compromise resilience and efficiency) but practical comparisons with existing SM schemes is limited. Further computational / communication comparisons would emphasise the practicality (and thus, adoptability) of the eSM scheme.