The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol

Joël Alwen[*]
Wickr Inc.
jalwen@wickr.com

Sandro Coretti[†]
New York University
corettis@nyu.edu

Yevgeniy Dodis[‡]
New York University
dodis@cs.nyu.edu

February 21, 2020

Abstract

Signal is a famous secure messaging protocol used by billions of people, by virtue of many secure text messaging applications including Signal itself, WhatsApp, Facebook Messenger, Skype, and Google Allo. At its core it uses the concept of "double ratcheting," where every message is encrypted and authenticated using a fresh symmetric key; it has many attractive properties, such as forward security, post-compromise security, and "immediate (no-delay) decryption," which had never been achieved in combination by prior messaging protocols.

While the formal analysis of the Signal protocol, and ratcheting in general, has attracted a lot of recent attention, we argue that none of the existing analyses is fully satisfactory. To address this problem, we give a clean and general definition of *secure messaging*, which clearly indicates the types of security we expect, including forward security, post-compromise security, and immediate decryption. We are the first to explicitly formalize and model the immediate decryption property, which implies (among other things) that parties seamlessly recover if a given message is permanently lost—a property not achieved by any of the recent "provable alternatives to Signal."

We build a modular "generalized Signal protocol" from the following components: (a) *continuous key agreement (CKA)*, a clean primitive we introduce and which can be easily and generically built from public-key encryption (not just Diffie-Hellman as is done in the current Signal protocol) and roughly models "public-key ratchets;" (b) *forward-secure authenticated encryption with associated data (FS-AEAD)*, which roughly captures "symmetric-key ratchets;" and (c) a two-input hash function that is a pseudorandom function (resp. generator with input) in its first (resp. second) input, which we term PRF-PRNG. As a result, in addition to instantiating our framework in a way resulting in the existing, widely-used Diffie-Hellman based Signal protocol, we can easily get post-quantum security and not rely on random oracles in the analysis.

We further show that our design can be elegantly extended to include other forms of "fine-grained state compromise" recently studied at CRYPTO'18, but *without sacrificing the immediate decryption property*. However, we argue that the additional security offered by these modifications is unlikely to justify the efficiency hit of using much heavier public-key cryptography in place of symmetric-key cryptography.

[*]Partially supported by the European Research Council under ERC Consolidator Grant (682815 - TOCNeT)
[†]Supported by NSF grants 1314568 and 1319051.
[‡]Partially supported by gifts from VMware Labs, Facebook and Google, and NSF grants 1314568, 1619158, 1815546.

Contents

1 Introduction

Signal [25] is a famous secure messaging protocol, which is—by virtue of many secure text messaging applications including Signal itself, WhatsApp [31], Facebook Messenger [12], Skype [23] and Google Allo [24]—used by billions of people. At its core it uses the concept of *double ratcheting*, where every message is encrypted and authenticated using a fresh symmetric key. Signal has many attractive properties, such as forward security and post-compromise security, and it supports immediate (no-delay) decryption. Prior to Signal's deployment, these properties had never been achieved in combination by messaging protocols.

Signal was designed by practitioners and was implemented and deployed well before any security analysis was obtained. In fact, a clean description of Signal has been posted by its inventors Marlinspike and Perrin [25] only recently. The write-up does an excellent job at describing the double-ratchet protocol, gives examples of how it is run, and provides security intuition for its building blocks. However, it lacks a formal definition of the secure messaging problem that the double ratchet solves, and, as a result, does not have a formal security proof.

A first formal treatment of Signal's security is provided in recent work by Cohn-Gordon *et al.* [7], who spell out numerous low-level details of Signal (including a number of enhancements required for a robust practical implementation). Their security definition, however, is quite involved and complex, and it is specifically targeted towards the existing Signal protocol—as opposed to the general secure messaging problem that Signal is trying to solve. Moreover, said paper does not define and address the immediate decryption feature of Signal, which is essential for justifying Signal's relative complexity compared to simpler protocols that do not have this feature.

Immediate decryption and its importance. One of the main issues any messaging scheme must address is the fact that messages might arrive out of order or be lost entirely. Additionally, parties can be offline for extended periods of time and send and receive messages asynchronously. Given these inherent constraints, immediate decryption is a very attractive feature. Informally, it ensures that when a legitimate message is (eventually) delivered to the recipient, the recipient can not only immediately decrypt the message but is also able to place it in the correct spot in relation to the other messages already received. Furthermore, immediate decryption also ensures an even more critical liveness property, termed *message-loss resilience (MLR)* in this work: if a message is permanently lost by the network, parties should still be able to communicate (perhaps realizing at some point that a message has never been delivered). Finally, even in settings where messages are eventually delivered (but could come out of order), giving up on immediate decryption seems cumbersome: should out-of-order messages be discarded or buffered by the recipient? If discarded, how will the sender (or the network) know that it should resend the message later? If buffered, how to prevent denial-of-service attacks and distinguish legitimate out-of-order messages (which cannot be immediately decrypted) from fake messages? While these questions could surely be answered (perhaps by making additional timing assumptions about the network), it appears that the simplest answer would be to design a secure messaging protocol which support immediate decryption. Indeed, to the best of our knowledge, all secure messaging services deployed in practice do have this feature (and, hence, MLR).

Additional properties. In practice, parties' states might occasionally leak. To address this concern, a secure messaging protocol should have the following two properties:

- **Forward secrecy (FS):** if the state of a party is leaked, none of the previous messages should get compromised (assuming they are erased from the state, of course).
- **Post-compromise security (PCS) (aka channel healing):** once the exposure of the party's state ends, security is restored after a few communication rounds.

In isolation, fulfilling either of these desirable properties is well understood: FS is achieved by using basic steam ciphers (aka pseudorandom generators (PRGs)) [3], while PCS [8] is achieved by some form of key agreement executed after the compromise, such as Diffie-Hellman. Unfortunately, these techniques, both of which involve some form of *key evolution*, are clearly at tension with immediate decryption when the network is fully asynchronous. Indeed, the main elegance of Signal, achieved by its *double-ratchet* algorithm, comes from the fact that FS and PCS are not only achieved together, but also *without sacrificing immediate decryption and MLR.*

Goals of this work. One of the main drawbacks of all formal Signal-related papers [2, 28, 18, 11], following the initial work of [7], is the fact that they all achieve FS and PCS by *explicitly giving up* not only on immediate decryption, but also MLR. (This is not merely a definitional issue as their constructions indeed cease any and all further functionality when, say, a single message is dropped in transit.) While such a drastic weakening of the liveness/correctness property considerably simplifies the algorithmic design for these provably secure alternatives to Signal, it also made them insufficient for settings where message loss is indeed possible. This can occur, in practice, due to a variety of reasons. For example, the protocol may be using an unreliable transport mechanism such as SMS or UDP. Alternatively, traffic may be routed (via more reliable TCP) through a central back-end server so as to facilitate asynchronous communication between end-points (as is very common for secure messaging deployments in practice). Yet, even in this setting, packet losses can still occur as the server itself may end up dropping messages due to a variety of unintended events such as due to outages or being subject to a heavy work/network load (say, because of an ongoing (D)DOS attack, partial outages, or worse yet, an emergency event generating sudden high volumes of traffic). With the goal of providing resilient communication even under these and similar realistic conditions, the main objectives of this work are to:

(a) propose formal definitions of *secure messaging* as a cryptographic primitive that *explicitly mandates immediate decryption and MLR*; and

(b) to provide an analysis of *Signal itself* in a well-defined *general* model for secure messaging.

Our work is the first to address either of these natural goals. Moreover, in order to improve the general understanding of secure messaging and to develop alternative (e.g., post-quantum secure) solutions, this paper aims at

(c) generalizing and abstracting out the reliance on the specific Diffie-Hellman key exchange (making the current protocol insecure in the post-quantum world) as well as clarifying the role of various cryptographic hash functions used inside the current Signal instantiation. That is, the idea is to build a "generalized Signal" protocol of which the current instantiation is a special case, but where other instantiations are possible, including those which are post-quantum secure and/or do not require random oracles.

1.1 Our Results

Addressing the points (a)-(c) above, this paper's main contributions are the following:

- Providing a clean definition of *secure messaging* that clearly indicates the expected types of security, including FS, PCS, and—for the first time—immediate decryption.
- Putting forth a modular *generalized* Signal protocol from several simpler building blocks:

  (1) *forward-secure authenticated encryption with associated data (FS-AEAD)*, which can be easily built from a regular PRG and AEAD and roughly models the so-called symmetric-key ratchet of Signal;

  (2) *continuous key agreement (CKA)*, which is a clean primitive that can easily be built generically from public-key encryption and roughly models the so-called public-key ratchet of Signal;

  (3) a two-input hash function, called *PRF-PRNG*, which is a pseudorandom function (resp. generator) in its first (resp. second) input and helps to "connect" the two ratchet types.
- Instantiating the framework such that we obtain the existing Diffie-Hellman-based protocol and observing that one can easily achieve post-quantum security (by using post-quantum-secure public-key encryption, such as [1, 5, 17]) and/or not rely on random oracles in the analysis.
- Extending the design to include other forms of "fine-grained state compromise" recently studied by Poettering and Rösler [28] and Jaeger and Stepanovs [18] but, once more, without sacrificing the immediate decryption property.

The secure messaging definition. The proposed secure messaging (SM) definition encompasses, in one clean game, (Figure 2) all desired properties, including FS as well as PCS and immediate decryption. The attacker in the definition is very powerful, has full control of the order of sending and receiving messages, can corrupt parties' state multiple times, and even controls the randomness used for encryption.[1] In order to avoid trivial and unpreventable attacks, a few restrictions need to be placed on an attacker $\mathcal{A}$. In broad strokes, the definition requires the following properties:

- When parties are uncompromised, i.e., when their respective states are unknown to $\mathcal{A}$, the protocol is secure against *active* attacks. In particular, the protocol must detect injected ciphertexts (not legitimately sent by one of the parties) and properly handle legitimate ciphertexts delivered in arbitrary order (capturing correctness and immediate decryption).
- When parties are uncompromised, messages are protected even against future compromise of either the sender or the receiver, modeling *forward security.*
- When one or both parties are compromised and the attacker remains *passive*, security is restored "quickly," i.e., within a few rounds of back-and-forth, which models *PCS.*

[1]Namely, good randomness is only needed to achieve PCS, while all other security properties hold even with the adversarially controlled randomness (when parties are not compromised).

While the proposed definition is still rather complex, we believe it to be intuitive and *considerably shorter and easier to understand* compared to the recent works of [18, 28], which are discussed in more detail in Section 1.2.

It should be stressed that the basic SM security in this paper only requires PCS against a passive attacker. Indeed, when an active attacker compromises the state of, say, party A, it can always send ciphertexts to the partner B in A's name (thereby even potentially hijacking A's communication with B and removing A from the channel altogether) or decrypt ciphertexts sent by B immediately following state compromise. As was observed by [18, 28] at CRYPTO'18, one might achieve certain limited forms of fine-grained security against active attacks. For example, it is not a priori clear if the attacker should be able to decrypt ciphertexts sent by A to B (if A uses good randomness) or forge legitimate messages from B to A (when A's state is exposed). We comment on these possible extensions in Section 6.3 but notice that they are still rather limited, given that the simple devastating attacks mentioned above are inherently non-preventable against active attackers immediately following state compromise. Thus, our main SM security notion simply *disallows all active attacks for Δ_{SM} epochs immediately following state compromise* where Δ_{SM} is the number of rounds of communication required to refresh a compromised state.

The building blocks. Since the original Signal protocol is quite subtle and somewhat tricky to understand, one of the main contributions of this work is to distill out three basic and intuitive building blocks used inside the double ratchet.

The first block is *forward-secure authenticated encryption with associated data (FS-AEAD)* and models secure messaging security inside a single so-called *epoch*; an epoch should be thought of as a unidirectional stream of messages sent by one of the parties, ending once a message from the other party is received. As indicated by the name, an FS-AEAD protocol must provide forward secrecy, but also immediate decryption. Capturing this makes the definition of FS-AEAD somewhat non-trivial (cf. Figure 5), but still simpler than that of general SM; in particular, no PCS is required (which allows us to define FS-AEAD as a deterministic primitive and not worry about poor randomness).

Building FS-AEAD turns out to be rather easy: in essence, one uses message counters as associated data for standard AEAD and a PRG to immediately refresh the secret key of AEAD after every message successfully sent or received. This is exactly what is done in Signal.

The second block is a primitive called *continuous key agreement (CKA)* (cf. Figure 3), which could be viewed as an abstraction of the DH-based public-key ratchet in Signal. CKA is a *synchronous* and *passive* primitive, i.e., parties A and B speak in turns, and no adversarial messages or traffic mauling are allowed. With each message sent or received, a party should output a fresh key such that (with "sending" keys generated by A being equal to "receiving" keys generated by B and vice versa). Moreover, CKA guarantees its own PCS, i.e., after a potential state exposure, security is restored within two rounds. Finally, CKA must be forward-secure, i.e., past keys must remain secure when the state is leaked. Forward security is governed by a parameter $\Delta_{\mathsf{CKA}} \geq 0$, which, informally, guarantees that all keys older than Δ_{CKA} rounds remain secure upon state compromise.

Not surprisingly, minimizing Δ_{CKA} results in faster PCS for secure messaging.[2] Fortunately, optimal CKA protocols achieving optimal $\Delta_{\mathsf{CKA}} = 0$ can be built generically from key-encapsulation mechanisms. Interestingly, the elegant DH-based CKA used by Signal achieves slightly sub-optimal $\Delta_{\mathsf{CKA}} = 1$, which is due to how long parties need to hold on to their secret exponents. However, the

[2]Specifically, the healing time of the generic Signal protocol presented in this work is $\Delta_{\mathsf{SM}} = 2 + \Delta_{\mathsf{CKA}}$.

Signal CKA saves about a factor of 2 in communication complexity, which makes it a reasonable trade-off in practice. We show that a similar tradeoff can also be achieved for schemes based on LWE. A general study of KEMs for which key generation and encapsulation can be merged has recently been put forth in follow-up work by Drucker and Gueron [10].

The third and final component of the generalized Signal protocol is a two-argument hash function P, called a PRF-PRNG, which is used to produce secret keys for FS-AEAD epochs from an entropy pool refreshed by CKA keys. More specifically, with each message exchanged using FS-AEAD, the parties try to run the CKA protocol "on the side," by putting the CKA messages as associated data. Due to asynchrony, the party will repeat a given CKA message until it receives a legitimate response from its partner, after which the CKA moves forward with the next message. Each new CKA key is absorbed into the state of the PRF-PRNG, which is then used to generate a new FS-AEAD key.

Informally, a PRF-PRNG takes as inputs a state σ and a CKA key I and produces a new state σ' and a (FS-AEAD) key k. It satisfies a *PRF* property saying that if σ is random, then $\mathsf{P}(\sigma, \cdot)$ acts like a PRF (keyed by σ) in that outputs (σ', k) on *adversarially chosen* inputs I are random. Moreover, it also acts like a PRNG in that, if the input I is random, then so are the resulting state σ' and key k. Observe that standard hash functions are assumed to satisfy this notion; alternatively, one can also very easily build a PRF-PRNG from any PRG and a pseudorandom permutation (cf. Section 4.3).

Generalized signal. Putting the above blocks together properly yields the generalized Signal protocol (cf. Figure 8). As a special case, one can easily obtain the existing Signal implementation[3] by using the standard way of building FS-AEAD from PRG and AEAD, CKA using the Diffie-Hellman based public-key ratchet mentioned above, and an appropriate cryptographic hash function in place of PRF-PRNG. However, many other variants become possible. For example, by using a generic CKA from DH-KEM, one may trade communication efficiency (worse by a factor of 2) for a shorter healing period Δ_{SM} (from 3 rounds to 2). More interestingly, using any post-quantum KEM, such as [1, 5, 17] results in a *post-quantum secure variant of Signal.* Finally, we also believe that our generalized double ratcheting scheme is much more intuitive than the existing DH-based variant, as it abstracts precisely the cryptographic primitives needed, including the two types ratchets, and what security is needed from each primitive.

Beyond double ratcheting to full signal. Following most of the prior (and concurrent) work [2, 28, 18, 11] (discussed in the next section), this paper primarily concerned with formalizing the double-ratchet aspect of the Signal protocol. This is certainly the most elegant and ingenious part of Signal, but it assumes that any set of two parties can correctly and securely agree on the initial secret key. The latter problem is rather non-trivial, especially (a) in the multi-user setting, when a party could be using a global public key to communicate with multiple recipients, some of which might be malicious, (b) when the initial secret key agreement is required to be non-interactive, and (c) when state compromise (including that of the master secret for the PKI) is possible, and even frequent. Some of those subtleties are discussed and analyzed by Cohn-Gordon *et al.* [7], but, once again, in a manner specific to the existing Signal protocol (rather than a general secure messaging primitive). Signal also suggests using the X3DH protocol [26] as one particular way to generate the

[3]For syntactic reasons having to do with our abstractions, our protocol is a minor variant of Signal, but is logically equivalent to Signal in every aspect.

initial shared key. Certainly, studying (and even appropriately defining) secure messaging *without idealized setup*, and analyzing "full Signal" in this setting, remains an important area for future research.

1.2 Related Work

The OTR (off-the-record) messaging protocol [4] is an influential predecessor of Signal, which was the first to introduce the idea of the DH-based double ratchet to derive fresh keys for each encrypted message. However, it was mainly suitable for synchronous back-and-fourth conversations, so Signal's double ratchet algorithm had to make a number of non-trivial modifications to extend the beautiful OTR idea into a full-fledged asynchronous messaging protocol.

Following the already discussed rigorous description of DH-based double ratcheting by Marlinspike and Perrin [25], and the protocol-specific analysis by Cohn-Gordon et al. [7], several formal analyses of ratcheting have recently appeared [2, 28, 18, 11]; they design definitions of various types of ratcheting and provide schemes meeting these definitions. As previously mentioned, all these works have the drawback of no longer satisfying immediate decryption.

Bellare et al. [2] looked at the question of *unidirectional ratcheting.* In this simplified variant of double (or bidirectional) ratcheting, the receiver is never corrupted, and never needs to update its state. Coupled with giving up immediate decryption, this allowed the authors to obtain a rather simple solution: in essence, generate fresh DH-key g^{x_i} (which was sent and authenticated as associated data of the previous message m_{i-1}) for each new message m_i, and perform (non-interactive) DH-exchange between g^{x_i} and the fixed receiver key g^y to derive the key for m_i. Unfortunately, extending this simple model (which does not appear to have direct applications) and solution to the case of bidirectional communication appeared non-trivial and was left to future work.

Bidirectionality has recently been achieved in two very similar work by Jaeger and Stepanovs [18] and Poettering and Rösler [28], both appearing at CRYPTO'18. The papers differ in syntax (one treats secure messaging while the other considers key exchange) and hence use different definitions. However, in spirit both papers attempt to model a bidirectional channel satisfying FS and PCS (but not immediate decryption). Moreover, both consider "fine-grained" PCS requirements which are not met by Signal's double ratchet protocol (and not required by the SM definition in this work). Namely, while Signal (and our work) do not offer any privacy and authenticity for messages immediately following state compromise (and until standard PCS kicks in), these papers demand "semi-security" even right after corrupting a party: privacy (but not authenticity) of *outgoing* messages and authenticity (but not privacy) of *incoming* messages. This semi-security is, no doubt, at least of theoretical interest. However, it seems unclear whether there are many realistic scenarios in which such a notion is sufficient. For example, one of the most important attacks not covered by Signal is when the attacker $\mathcal{A}$ learns A's secret state and can easily disrupt her session with B by sending a fake message to either party before parties would normally heal by PCS. Fine-grained PCS will prevent $\mathcal{A}$ from sending a fake message from B to A, but (inherently!) cannot prevent $\mathcal{A}$ from sending a fake message from A to B. However, either of these options is equally effective at disrupting the channel, so it is not clear how fine-grained PCS helps in this very important attack scenario. In principle this might not be that much of a concern except that the extra semi-security appears to come at a steep price: both papers use growing (and potentially unbounded) state as well as heavy techniques from public-key cryptography, including certain types of "key-updatable" public-key signature and encryption schemes; all known constructions of such key-updatable public-key encryption rely onierarchical identity-based encryption [13] (HIBE), which is extremely inefficient compared to primitives used by Signal (nor are HIBEs even nearly as widely and robustly implemented). Finally, the formal definitions given by both papers are quite involved and hard to parse, having lots of variables and many lines of pseudocode, making it difficult to work with when analyzing new schemes. Overall, we feel that assuming a passive attacker for few messages before *full* PCS would kick in seems to be a good compromise between preventing many realistic attacks, while keeping the resulting schemes and definitions practical, efficient and with a limited amount of implementation overhead. More discussion can be found in Section 6.1, including a practical compromise: an (informally stated) extension to Signal which achieves a slightly weaker form of fine-grained compromise than [28, 18], yet still using only constant sized states, bandwidth and computation as well as comparatively lightweight (and widely implemented) public-key primitives (i.e., regular signature and public-key encryption).

Finally, the notion of immediate decryption is reminiscent in spirit to the zero round trip time (0-RTT) communication with forward secrecy which was recently studied by [15, 9]. However, the latter primitive is stateless on the sender side, making it more difficult to achieve (e.g., the schemes of [15, 9] use a heavy tool called *puncturable encryption* [14]).

Concurrent and Independent Work. We have recently become aware of two concurrent and independent works by Durak and Vaudenay [11] and Jost, Maurer and Mularczyk [19]. Like other prior works, these works (1) designed their own protocols and did not analyze Signal; and (2) do not satisfy immediate decryption or even message-loss resilience (in fact, they critically rely on receiving messages from one party in order). Both works also provide formal notions of security, including privacy, authenticity, and a new property called unrecoverability by [11] and post-impersonation authentication by [19]: if an active attacker sends a fake message to the recipient immediately following state compromise of the sender, the sender can, by design, never recover (and, thus, will notice the attack by being unable to continue the conversation). We find this property very interesting and briefly comment our initial thoughts on it in Section 6.3. We also notice that the work of [11] requires computation linear in the number of messages within an epoch (both for sending and for receiving); moreover, this linear computation involves *public-key* operations (in fact, no symmetric keys are used at all). The work of [19] also uses public-key operations for every message sent, even within a single epochs. Hence, our generalized Signal protocol appears to be much more efficient than the solutions of [11, 19].

However, a fairer comparison will be to look at the public-key variant of Signal that we sketch in Section 6.3, since this is when we achieve certain forms of "fine-grained compromise" also achieved by [11, 19]. We find our protocol to be comparable in efficiency with that of [19] and still more efficient than [11] (unless epochs are guaranteed to be short). Security-wise, putting aside immediate decryption (not modeled/achieved by [11, 19]) and unrecoverability (not modeled/achieved by Signal and this work), it appears that our work and the works of [11, 19] achieve somewhat comparable other properties, including forward security and PCS.

2 Preliminaries

2.1 Game-Based Security and Notation

All security definitions in this work are game-based, i.e., they consider games executed between a challenger and an adversary. The games have one of the following formats:

- **Unpredictability games:** First, the challenger executes the special init procedure, which sets up the game. Subsequently, the attacker is given access to a set of oracles that allow it to interact with the scheme in question. The goal of the adversary is to provoke a particular, game-specific *winning* condition. The *advantage* of an adversary $\mathcal{A}$ against construction C in an unpredictability game Γ^C is
$$\mathrm{Adv}_\Gamma^C(\mathcal{A}) \ := \ \mathsf{P}[\mathcal{A} \text{ wins } \Gamma^C] \ .$$
- **Indistinguishability games:** In addition to setting up the game, the init procedure samples a secret bit $b \in \{0,1\}$. The goal of the adversary is to determine the value of b. Once more, upon completion of init, the attacker interacts arbitrarily with all available oracles up to the point where it outputs a guess bit b'. The adversary *wins* the game if $b = b'$. The *advantage* of an adversary $\mathcal{A}$ against construction C in an indistinguishability game Γ is
$$\mathrm{Adv}_\Gamma^C(\mathcal{A}) \ := \ 2 \cdot \left|\mathsf{P}[\mathcal{A} \text{ wins } \Gamma^C] - 1/2\right| \ .$$

With the above in mind, to describe a any security (or correctness) notion, one need only specify the init oracle and the oracles available to $\mathcal{A}$. The following special keywords are used to simplify the exposition of the security games:

- **req** is followed by a condition; if the condition is not satisfied, the oracle/procedure containing the keyword is exited and all actions by it are undone.
- **win** is used to declare that the attacker has won the game; it can be used for both types of games above.
- **end** disables all oracles and returns all values following it to the attacker.

Moreover, the descriptions of some games/schemes involve *dictionaries*. For ease of notation, these dictionaries are described with the *array-notation* described next, but it is important to note that they are to be implemented by a data structure whose size grows (linearly) with the number of elements *in* the dictionary (unlike arrays):

- **Initialization:** The statement $D[\cdot] \leftarrow \lambda$ initializes an *empty* dictionary D.
- **Adding elements:** The statement $D[i] \leftarrow v$ adds a value v to dictionary D with key i, overriding the value previously stored with key i if necessary.
- **Retrieval:** The expression $D[i]$ returns the value v with key i in the dictionary; if there are no values with key i, the value λ is returned.
- **Deletion:** The statement $D[i] \leftarrow \lambda$ *deletes* the value v corresponding to key i.

Finally, sometimes the random coins of certain probabilistic algorithms are made explicit. For example, $y \leftarrow A(x; r)$ means that A, on input x and with random tape r, produces output y. If r is not explicitly stated, it is assumed to be chosen uniformly at random; in this case, the notation $y \leftarrow\!\!\$\ A(x)$ is used.

Oracles for AEAD Game

```
init
    $K \leftarrow \mathcal{K}$
    $e^* \leftarrow \lambda$
    $b \leftarrow \{0,1\}$

encrypt $(a, m)$
    if $b = 0$
        $e^* \leftarrow \mathsf{Enc}(K, a, m)$
    else
        $e^* \leftarrow \mathcal{C}$
    return $e^*$

decrypt $(a, e)$
    if $e = e^*$ or $b = 1$
        return $\perp$
    return $\mathsf{Dec}(K, a, e)$
```

Figure 1: *Oracles of the IND-CCA security game for an AEAD scheme* $(\mathsf{Enc}, \mathsf{Dec})$*, where* **encrypt** *is a one-time oracle.*

2.2 Authenticated Encryption

Definition 1. *An* authenticated encryption with associated data (AEAD) scheme *is a pair of algorithms* $\mathsf{AE} = (\mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

- ***Encryption:*** Enc *takes a key* K*, associated data* a*, and a message* m *and produces a ciphertext* $e \leftarrow \mathsf{Enc}(K, a, m)$*.*
- ***Decryption:*** Dec *takes a key* K*, associated data* a*, and a ciphertext* e *and produces a message* $m \leftarrow \mathsf{Dec}(K, a, e)$*.*

All AEAD schemes in this paper are assumed to be deterministic, i.e., all randomness stems from the key K.

Correctness. An AEAD scheme is *correct* if for all keys K and all pairs (K, a),

$$\mathsf{Dec}(K, a, \mathsf{Enc}(K, a, m)) \ = \ m \, .$$

Security. In order to be used in the constructions in this paper, AEAD schemes need only satisfy one-time IND-CCA security. This is captured by the game depicted in Figure 1. It provides access to a one-time encryption oracle that returns either an encryption of the message specified by the attacker or a random ciphertext from the space $\mathcal{C}$ of all ciphertexts, depending on a randomly chosen bit b. Moreover, the attacker may query a decryption oracle arbitrarily many times (except on the challenge ciphertext), which, however, always returns $\perp$ if $b = 1$. The advantage of an adversary $\mathcal{A}$ attacking an AEAD scheme AE is denoted by $\mathrm{Adv}^{\mathsf{AE}}_{\text{ot-cca}}(\mathcal{A})$; the attacker is parametrized by its running time t.

Definition 2. *An AEAD scheme* AE *is* (t, ε)-one-time-CCA-secure *if for all* t*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{AE}}_{\text{ot-cca}}(\mathcal{A}) \ \leq \ \varepsilon \ .$$

2.3 Key-Encapsulation Mechanism

Definition 3. *A* key-encapsulation mechanism *(KEM) is a public-key primitive consisting of three algorithms* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

- ***Key generation:*** Gen *takes a (implicit) security parameter and outputs a fresh key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{Gen}$.
- ***Encapsulation:*** Enc *takes a public key* pk *and produces a ciphertext and a symmetric key* $(c, k) \leftarrow\$ \mathsf{Enc}(\mathsf{pk})$.
- ***Decapsulation:*** Dec *takes a secret key* sk *and a ciphertext* c *and recovers the symmetric key* $k \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$.

Correctness. A KEM must satisfy the following standard correctness property:

$$\mathsf{P}[(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{Gen}, (c, k) \leftarrow\$ \mathsf{Enc}(\mathsf{pk}), k' \leftarrow \mathsf{Dec}(\mathsf{sk}, c) : k = k'] \;=\; 1 \;.$$

Security. KEMs are required to satisfy IND-CPA security, which asks that no adversary be able to tell a random symmetric key from an encapsulated one (without the decapsulation key). This is captured by considering the following game: it chooses a random bit $b \leftarrow \{0, 1\}$, produces a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}$, and generates $(c, k_0) \leftarrow \mathsf{Enc}(\mathsf{pk})$ as well as a uniform random key k_1; an attacker $\mathcal{A}$ wins the game if it correctly guesses b, given pk, c, and k_b. The advantage of $\mathcal{A}$ is denoted by $\mathrm{Adv}^{\mathsf{KEM}}_{\mathrm{cpa}}(\mathcal{A})$; the attacker is parametrized by its running time t.

Definition 4. *A key-encapsulation mechanism* KEM *is* (t, ε)-CPA-secure *if for all* t*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{KEM}}_{\mathrm{cpa}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

Section C of the appendix briefly recapitulates the standard El-Gamal KEM based on DDH and the Frodo KEM [6], which is based on learning with errors (LWE).

2.4 Pseudorandom Generators

A *pseudorandom generator (PRG)* is a function $G : \mathcal{W} \to \mathcal{W} \times \mathcal{K}$ such that $G(U)$ is indistinguishable from U' for uniformly random $U \in \mathcal{W}$ and $U' \in \mathcal{W} \times \mathcal{K}$. The advantage of an attacker $\mathcal{A}$ at distinguishing between these two distributions is denoted by $\mathrm{Adv}^{G}_{\mathrm{prg}}(\mathcal{A})$; the attacker is parametrized by its running time t.

Definition 5. *A pseudorandom generator* G *is* (t, ε)-secure *if for all* t*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{G}_{\mathrm{prg}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

3 Secure Messaging

A *secure messaging (SM)* scheme allows two parties A and B to communicate securely bidirectionally and is expected to satisfy the following informal requirements:

- **Correctness:** If no attacker interferes with the transmission, B outputs the messages sent by A in the correct order and vice versa.
- **Immediate decryption and message-loss resilience (MLR):** Messages must be decrypted as soon as they arrive and may not be buffered; if a message is lost, the parties do not stall.

- **Authenticity:** While the parties' states are uncompromised (i.e., unknown to the attacker), the attacker cannot change the messages sent by them or inject new ones.
- **Privacy:** While the parties' states are uncompromised, an attacker obtains no information about the messages sent.
- **Forward secrecy (FS):** All messages sent and received prior to a state compromise of either party (or both) remain hidden to an attacker.
- **Post-compromise security (PCS, aka "healing"):** If the attacker remains passive (i.e., does not inject any corrupt messages), the parties recover from a state compromise (assuming each has access to fresh randomness).
- **Randomness leakage/failures:** While the parties' states are uncompromised, all the security properties above except PCS hold even if the attacker completely controls the parties' local randomness. That is, good randomness is only required for PCS.

This section presents the syntax of and a formal security notion for SM schemes.

3.1 Syntax

Formally, an SM scheme consists of two initialization algorithms, which are given an initial shared key k, as well as a sending algorithm and a receiving algorithm, both of which keep (shared) state across invocations. The receiving algorithm also outputs a so-called epoch number and an index, which can be used to determine the order in which the sending party transmitted their messages.

Definition 6. *A* secure-messaging (SM) scheme *consists of four probabilistic algorithms* $\mathsf{SM} = (\mathsf{Init\text{-}A}, \mathsf{Init\text{-}B}, \mathsf{Send}, \mathsf{Rcv})$*, where*

- $\mathsf{Init\text{-}A}$ *(and similarly* $\mathsf{Init\text{-}B}$*) takes a key* k *and outputs a state* $s_{\mathsf{A}} \leftarrow \mathsf{Init\text{-}A}(k)$*,*
- Send *takes a state* s *and a message* m *and produces a new state and a ciphertext* $(s', c) \leftarrow\$ \mathsf{Send}(s, m)$*, and*
- Rcv *takes a state* s *and a ciphertext* c *and produces a new state, an epoch number, an index, and a message* $(s', t, i, m) \leftarrow \mathsf{Rcv}(s, c)$*.*

3.2 Security

3.2.1 Basics

The security notion for SM schemes considered in this paper is intuitive in principle. However, formalizing it is non-trivial and somewhat cumbersome due to a number of subtleties that naturally arise and cannot be avoided if the criteria put forth at the beginning of Section 3 are to be met. Therefore, before presenting the definition itself, this section introduces some basic concepts that will facilitate understanding of the definition.

Epochs. SM schemes proceed in so-called *epochs*, which roughly correspond the "back-and-forth" between the two parties A and B. By convention, odd epoch numbers t are associated with A sending and B receiving, and the other way around for even epochs. Note, however, that SM schemes are completely asynchronous, and, hence, epochs overlap to a certain extent. Correspondingly, consider two epoch counters t_{A} and t_{B} for A and B, respectively, satisfying the following properties:

- The two counters are never more than one epoch apart, i.e., $|t_{\mathsf{A}} - t_{\mathsf{B}}| \leq 1$ at all times.
- When A receives an epoch-t message from B for $t = t_{\mathsf{A}} + 1$, it sets $t_{\mathsf{A}} \leftarrow t$ (even). The next time A sends a message, t_{A} is incremented again (to an odd value).
- Similarly, when B receives an epoch-t message from A for $t = t_{\mathsf{B}} + 1$, it sets $t_{\mathsf{B}} \leftarrow t$ (odd). The next time B sends a message, t_{B} is incremented again (to an even value).

Message indices. Within an epoch, messages are identified by a simple counter. To capture the property of immediate decryption and MLR, the receive algorithm of an SM scheme is required to output the correct epoch number and index *immediately* upon reception of a ciphertext, even when messages arrive out of order.

Corruptions and their consequences. Since SM schemes are required to be forward-secure and to recover from state compromise, any SM security game must allow the attacker to learn the state of either party at any given time. Moreover, to capture authenticity and privacy, the attacker should be given the power to inject malicious ciphertexts and to call a (say) left-or-right challenge oracle, respectively. These requirements, however, interfere as follows:

- When either party is in a compromised state, the attacker cannot invoke the challenge oracle since this would allow him to trivially distinguish.
- When either party is in a compromised state, the attacker can trivially forge ciphertexts and must therefore be barred from calling the inject oracle.
- When the receiver of messages in transmission is compromised, these messages lose all security, i.e., the attacker learns their content and can replace them by a valid forgery. Consequently, while any challenge ciphertext is in transmission, the recipient may not be corrupted. Similarly, an SM scheme must be able to deal with forgeries of compromised messages (once the parties have healed).

These issues require that the security definition keep track of ciphertexts *in transmission*, of *challenge* ciphertexts, and of *compromised* ciphertexts; this will involve some (slightly cumbersome) record keeping.

Natural SM schemes. For simplicity, SM schemes in this work are assumed to satisfy the natural requirements below.[4]

Definition 7. *An SM scheme* $\mathsf{SM} = (\mathsf{Init\text{-}A}, \mathsf{Init\text{-}B}, \mathsf{Send}, \mathsf{Rcv})$ *is* natural *if the following criteria are satisfied:*

(A) Whenever Rcv *outputs* $m = \perp$*, the state remains unchanged.*

(B) Any given ciphertext corresponds to an epoch t *and an index* i*, i.e., the values* (t, i) *output by* Rcv *are an (efficiently computable) function of* c*.*

(C) Algorithm Rcv *never accepts two messages corresponding to the same pair* (t, i)*.*

[4]The reader may skip over this definition on first read. The properties are referenced where they are needed.

(D) A party always rejects ciphertexts corresponding to an epoch in which the party does not act as receiver

(E) If a party, say A*, accepts a ciphertext corresponding to an epoch* t*, then* $t_{\mathsf{A}} \geq t - 1$.

3.2.2 The Security Game

The security game, which is depicted in Figure 2, consists of an initialization procedure **init** and of

- two "send" oracles, **transmit-A** (normal transmission) and **chall-A** (challenge transmission);
- two "receive" oracles, **deliver-A** (honest delivery) and **inject-A** (for forged ciphertexts); and
- a corrupt oracle **corr-A**

pertaining to party A, and of the corresponding oracles pertaining to B. Moreover, Figure 2 also features an epoch-management function **ep-mgmt**, a function **sam-if-nec** explained below, and two record-keeping functions **record** and **delete**; these functions cannot be called by the attacker. The game is parametrized by Δ_{SM}, which relates to how fast parties recover from a state compromise. All components are explained in detail below, following the intuition laid out above.

The advantage of $\mathcal{A}$ against an SM scheme SM is denoted by $\mathrm{Adv}^{\mathsf{SM}}_{\mathrm{sm},\Delta_{\mathsf{SM}}}(\mathcal{A})$. The attacker is parameterized by its running time t, the total number of queries q it makes, and the maximum number of epochs q_{ep} it runs for.

Definition 8. *A secure-messaging scheme* SM *is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$-secure *if for all* (t, q, q_{ep})*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{SM}}_{\mathrm{sm},\Delta_{\mathsf{SM}}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

Initialization and state. The initialization procedure chooses a random key and initializes the states s_{A} and s_{B} of A and B, respectively. Moreover, it defines several variables to keep track of the execution: (1) t_{A} and t_{B} are the epoch counters for A and B, respectively; (2) variables i_{A} and i_{B} count how many messages have been sent by each party in their respective current epochs; (3) t_{L} records the last time either party's state was leaked to the attacker and is used, together with t_{A} and t_{B}, to preclude trivial attacks; (4) the sets trans, chall, and comp will contain records and allow to track ciphertexts in transmission, challenge ciphertexts, and compromised ciphertexts, respectively; (5) the bit b is used to create the challenge.

Sampling if necessary. The send oracles **transmit-A** and **chall-A** allow the attacker to possibly control the random coins r of Send. If $r = \perp$, the function samples $r \leftarrow\$ \mathcal{R}$ (from some appropriate set $\mathcal{R}$), and returns (r, good), where good indicates that fresh randomness is used. If, on the other hand, $r \neq \perp$, the function returns (r, bad), indicating, via bad, that adversarially controlled randomness is used.

Epoch management. The epoch management function **ep-mgmt** advances the epoch of the calling party if that party's epoch counter has a "receiving value" (even for A; odd for B) and resets the index counter. The flag argument is to indicate whether fresh or adversarial randomness is used. If a currently corrupted party starts a new epoch with bad randomness, the new epoch is considered corrupted. However, if it does not start a new epoch, bad randomness does not make

Security Game for Secure Messaging

```
init
    $k \leftarrow\!\$\ \mathcal{K}$
    $s_{\mathsf{A}} \leftarrow \mathsf{Init\text{-}A}(k)$
    $s_{\mathsf{B}} \leftarrow \mathsf{Init\text{-}B}(k)$
    $(t_{\mathsf{A}}, t_{\mathsf{B}}) \leftarrow (0, 0)$
    $i_{\mathsf{A}}, i_{\mathsf{B}} \leftarrow 0$
    $t_{\mathsf{L}} \leftarrow -\infty$
    $\mathsf{trans}, \mathsf{chall}, \mathsf{comp} \leftarrow \emptyset$
    $b \leftarrow\!\$\ \{0, 1\}$

corr-A
    req $\mathsf{B} \notin \mathsf{chall}$
    $\mathsf{comp} \overset{+}{\leftarrow} \mathsf{trans}(\mathsf{B})$
    $t_{\mathsf{L}} \leftarrow \max(t_{\mathsf{A}}, t_{\mathsf{B}})$
    return $s_{\mathsf{A}}$

transmit-A $(m, r)$
    $(r, \mathsf{flag}) \leftarrow$ sam-if-nec$(r)$
    ep-mgmt$(\mathsf{A}, \mathsf{flag})$
    $i_{\mathsf{A}}$ ++
    $(s_{\mathsf{A}}, c) \leftarrow \mathsf{Send}(s_{\mathsf{A}}, m; r)$
    record$(\mathsf{A}, \mathsf{norm}, m, c)$
    return $c$

chall-A $(m_0, m_1, r)$
    $(r, \mathsf{flag}) \leftarrow$ sam-if-nec$(r)$
    ep-mgmt$(\mathsf{A}, \mathsf{flag})$
    req $\mathsf{safe\text{-}ch_A}$ and
      $|m_0| = |m_1|$
    $i_{\mathsf{A}}$ ++
    $(s_{\mathsf{A}}, c) \leftarrow \mathsf{Send}(m_b; r)$
    record$(\mathsf{A}, \mathsf{chall}, m_b, c)$
    return $c$

deliver-A $(c)$
    req $(\mathsf{B}, t, i, m, c) \in \mathsf{trans}$
        for some $t, i, m$
    $(s_{\mathsf{A}}, t', i', m') \leftarrow \mathsf{Rcv}(s_{\mathsf{A}}, c)$
    if $(t', i', m') \neq (t, i, m)$
        win
    if $(t, i, m) \in \mathsf{chall}$
        $m' \leftarrow \bot$
    $t_{\mathsf{A}} \leftarrow \max(t_{\mathsf{A}}, t)$
    delete$(t, i)$
    return $(t', i', m')$

inject-A $(c)$
    req $(\mathsf{B}, c) \notin \mathsf{trans}$ and $\mathsf{safe\text{-}inj}$
    $(s_{\mathsf{A}}, t', i', m') \leftarrow \mathsf{Rcv}(s_{\mathsf{A}}, c)$
    if $m' \neq \bot$ and $(\mathsf{B}, t', i') \notin \mathsf{comp}$
        win
    $t_{\mathsf{A}} \leftarrow \max(t_{\mathsf{A}}, t')$
    delete$(t', i')$
    return $(t', i', m')$
```

```
ep-mgmt $(\mathsf{P}, \mathsf{flag})$
    if $\mathsf{P} = \mathsf{A}$ and $t_{\mathsf{P}}$ even or
       $\mathsf{P} = \mathsf{B}$ and $t_{\mathsf{P}}$ odd
        if $\mathsf{flag} = \mathsf{bad}$ and
           $\neg\mathsf{safe\text{-}ch_P}$
            $t_{\mathsf{L}} \leftarrow t_{\mathsf{P}} + 1$
        $t_{\mathsf{P}}$ ++
        $i_{\mathsf{P}} \leftarrow 0$

sam-if-nec $(r)$
    $\mathsf{flag} \leftarrow \mathsf{bad}$
    if $r = \bot$
        $r \leftarrow\!\$\ \mathcal{R}$
        $\mathsf{flag} \leftarrow \mathsf{good}$
    return $(r, \mathsf{flag})$

record $(\mathsf{P}, \mathsf{flag}, m, c)$
    $\mathsf{rec} \leftarrow (\mathsf{P}, t_{\mathsf{P}}, i_{\mathsf{P}}, m, c)$
    $\mathsf{trans} \overset{+}{\leftarrow} \mathsf{rec}$
    if $\neg\mathsf{safe\text{-}ch_P}$
        $\mathsf{comp} \overset{+}{\leftarrow} \mathsf{rec}$
    if $\mathsf{flag} = \mathsf{chall}$
        $\mathsf{chall} \overset{+}{\leftarrow} \mathsf{rec}$

delete $(t, i)$
    $\mathsf{rec} \leftarrow (\mathsf{P}, t, i, m, c)$
          for some $\mathsf{P}, m, c$
    $\mathsf{trans}, \mathsf{chall}, \mathsf{comp} \overset{-}{\leftarrow} \mathsf{rec}$

$\mathsf{safe\text{-}ch_P}$ $:\Longleftrightarrow$ $t_{\mathsf{P}} \geq t_{\mathsf{L}} + \Delta_{\mathsf{SM}}$

$\mathsf{safe\text{-}inj}$
 $:\Longleftrightarrow$ $\min(t_{\mathsf{A}}, t_{\mathsf{B}}) \geq t_{\mathsf{L}} + \Delta_{\mathsf{SM}}$
```

Figure 2: *Oracles corresponding to party* A *of the SM security game for a scheme* $\mathsf{SM} = (\mathsf{Init\text{-}A}, \mathsf{Init\text{-}B}, \mathsf{Send}, \mathsf{Rcv})$*; the oracles for* B *are defined analogously.*

the ciphertext corrupted. This captures that randomness should only be used for PCS (but for none of the other properties mentioned above).

Record keeping. The game keeps track of ciphertexts in transmission, of challenge ciphertexts, and of compromised ciphertexts. Records have the format $(\mathsf{P}, t_{\mathsf{P}}, i_{\mathsf{P}}, m, c)$, where P is the sender, t_{P} the epoch in which the message was sent, i_{P} the index within the epoch, m the message itself, and c the ciphertext.

Whenever **record** is called, the new record is added to the set trans. If a party is not in a safe state, the record is also added to the set of compromised ciphertexts comp. If the function is called with $\mathsf{flag} = \mathsf{chall}$, the record is added to chall. The function **delete** takes an epoch number and an index and removes the corresponding record from all three record keeping sets trans, chall, and comp.

Sometimes, it is convenient to refer to a particular record (or a set thereof) by only specifying parts of it. For example, the expression $\mathsf{B} \notin \mathsf{chall}$ is equivalent to there not being any record $(\mathsf{B}, *, *, *, *)$ in the set chall. Similarly, $\mathsf{trans}(\mathsf{B})$ is the set of all records of this type in trans.

Send oracles. Both send oracles, **transmit-A** and **chall-A**, begin with **sam-if-nec**, which samples fresh randomness if necessary, followed by a call to **ep-mgmt**. Observe that the flag argument is set to $\mathsf{flag} \leftarrow \mathsf{good}$ by **sam-if-nec** if fresh randomness is used, and to $\mathsf{flag} \leftarrow \mathsf{bad}$ otherwise. Subsequently:

- **transmit-A** increments i_{A}, executes Send, and creates a record using $\mathsf{flag} = \mathsf{norm}$, indicating that this is not a challenge ciphertext. Observe that if A is not currently in a safe state, the record is added to comp.
- **chall-A** works similarly to **transmit-A**, except that one of the two inputs is selected according to b, and the record is saved with $\mathsf{flag} = \mathsf{chall}$, which will cause it to be added to the challenges chall. Note that **chall-A** can only be called when A is not in a compromised state, which is captured by the statement **req** $\mathsf{safe\text{-}ch}_{\mathsf{A}}$.

The oracles for B are defined analogously.

Receive oracles. Two oracles are available by which the attacker can get A to receive a ciphertext: **deliver-A** is intended for honest delivery, i.e., to deliver ciphertexts created by B, whereas **inject-A** is used to inject forgeries. These rules are enforced by checking (via **req**) the set trans.

- **deliver-A**: The ciphertext is first passed through Rcv, which must correctly identify the values t, i, and m recorded when c was created; if it fails to do so, the correctness property is violated and the attacker immediately wins the game. In case c was a challenge, the decrypted message is replaced by $\perp$ in order to avoid trivial attacks. Before returning the output of Rcv, t_{A} is incremented if t is larger than t_{A}, and the record corresponding to c is deleted.
- **inject-A**: Again, the ciphertext is first passed through Rcv. Unless the ciphertext corresponds[5] to $(t, i) \in \mathsf{comp}$, algorithm Rcv must reject it; otherwise, authenticity is violated and the attacker wins the game. The final instructions are as in **deliver-A**. Oracle **inject-A** may only be called if neither party is currently recovering from state compromise, which is taken care of by flag $\mathsf{safe\text{-}inj}$.

The oracles for B are defined analogously.

By deleting records at the end of **deliver-A** and **inject-A**, the game enforces that no replay attacks take place. For example, if a ciphertext c that at some point is in trans is accepted twice,

[5]cf. Property (B) in Definition 7.

the second time counts as a forgery. Similarly, if two forgeries for a compromised pair (t, i) are accepted, the attacker wins as well. Note, however, that natural schemes do not allow replay attacks (cf. Property (C) in Definition 7).

Corruption oracles. The corruption oracle for A, **corr-A**, can be called whenever no challenges are in transit from B to A, i.e., when $\mathsf{B} \notin \mathsf{chall}$. If corruption is allowed, all ciphertexts in transit sent by B become compromised. Before returning A's state, the oracle updates the time of the most recent corruption. The corruption oracle **chall-B** for B is defined similarly.

3.3 Simplified Properties

Proving SM security is facilitated by considering the three simplified properties *correctness*, *authenticity*, and *privacy*. These properties together imply full SM security, as stated at the end of this section. All three of them are obtained by modifying the original SM game (cf. Figure 2).

Correctness. For the correctness game, there are no oracles **chall-A** and **chall-B**, and only *reduced* inject oracles are available, which differ from the normal ones as follows:

- if ciphertext c does not correspond to $(t', i') \in \mathsf{comp}$, **inject-A** and **inject-B** immediately return $(t', i', \perp)$;[6]
- the **if**-clause is removed completely.

Essentially, the reduced inject oracle only processes ciphertexts corresponding to $(t, i) \in \mathsf{comp}$, which is unavoidable even after healing since the attacker knows the keys with which to decrypt and authenticate these messages. Observe that the only way to win the game is via the **win** instruction in **deliver-A** or **deliver-B**. The advantage of $\mathcal{A}$ in the correctness game is denoted by $\mathrm{Adv}^{\mathsf{SM}}_{\mathrm{corr},\Delta_{\mathsf{SM}}}(\mathcal{A})$.

Definition 9. *A secure-messaging scheme* SM *is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$-correct *if for all* (t, q, q_{ep})*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{SM}}_{\mathrm{corr},\Delta_{\mathsf{SM}}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

Authenticity. In the authenticity variant of the SM game, there are no oracles **chall-A** and **chall-B**, and the **win**-instruction inside **deliver-A** and **deliver-B** is removed. Furthermore, the attacker must call **init** with two arguments t^*_{L} and t^* with $t^* \geq t^*_{\mathsf{L}} + \Delta_{\mathsf{SM}}$, where t^* is the epoch it is trying to attack and t^*_{L} is the last corruption event before the attempt. To formally define this, consider the case where t^* is *even*, i.e., B is the sender and A the receiver in the epoch under attack; the case where t^* is odd works analogously. Then, the authenticity game additionally differs from the original as follows:

- if at any point $t_{\mathsf{L}} \in \{t^*_{\mathsf{L}} + 1, \ldots, t^* - 1\}$, the attacker loses the game immediately;
- if A (the receiver in epoch t^*) is corrupted any time once $t_{\mathsf{A}} > t^*_{\mathsf{L}}$, the attacker loses the game immediately;

[6]Recall that, by virtue of Property (B) in Definition 7, (t', i') output by Rcv are assumed to be efficiently deducible from a ciphertext c.

- the inject oracles are reduced (see above) *except* for ciphertexts corresponding to epoch t^*.

The advantage of $\mathcal{A}$ in the authenticity game is denoted by $\text{Adv}^{\mathsf{SM}}_{\text{auth},\Delta_{\mathsf{SM}}}(\mathcal{A})$.

Definition 10. *A secure-messaging scheme* SM *is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$-authentic *if for all* (t, q, q_{ep})-*attackers* $\mathcal{A}$*,*

$$\text{Adv}^{\mathsf{SM}}_{\text{auth},\Delta_{\mathsf{SM}}}(\mathcal{A}) \ \leq \ \varepsilon \ .$$

Privacy. The privacy variant only provides *reduced* inject oracles (see above), and the **if**-clause with the **win** instruction in **deliver-A** is removed completely. Furthermore, the attacker must call **init** with two arguments t^*_{L} and t^* with $t^* \geq t^*_{\mathsf{L}} + \Delta_{\mathsf{SM}}$, where t^* is the epoch it is trying to attack and t^*_{L} is the last corruption event before the attempt. Formally, this is enforced as follows:

- if at any point $t_{\mathsf{L}} \in \{t^*_{\mathsf{L}} + 1, \ldots, t^* - 1\}$, the attacker loses the game immediately (i.e., the outcome of the game is a uniformly random bit);
- the challenge oracle for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ has an additional $t_{\mathsf{P}} = t^*$ constraint in the **req**-statement.

Observe that the condition $t_{\mathsf{P}} = t^*$ effectively means that only one challenge oracle is available (**chall-A** if t^* is even; **chall-B** if t^* is odd). The advantage of $\mathcal{A}$ in the privacy game denoted by $\text{Adv}^{\mathsf{SM}}_{\text{priv},\Delta_{\mathsf{SM}}}(\mathcal{A})$.

Definition 11. *A secure-messaging scheme* SM *is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$-private *if for all* (t, q, q_{ep})-*attackers* $\mathcal{A}$*,*

$$\text{Adv}^{\mathsf{SM}}_{\text{auth},\Delta_{\mathsf{SM}}}(\mathcal{A}) \ \leq \ \varepsilon \ .$$

Combining the properties. The following theorem relates the simplified properties to the full SM security definition; its proof can be found in Appendix A.

Theorem 1. *Assume an SM scheme* SM *is*

- $(t', q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon_{\text{corr}})$*-correct,*
- $(t', q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon_{\text{auth}})$*-authentic, and*
- $(t', q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon_{\text{priv}})$*-private.*

Then, it is also $(t, \Delta_{\mathsf{SM}}, q, q_{\mathsf{ep}}, \varepsilon)$*-secure, where*

$$\varepsilon \ \leq \ \varepsilon_{\text{corr}} + q_{\mathsf{ep}}^2 \cdot (\varepsilon_{\text{auth}} + \varepsilon_{\text{priv}}) \ .$$

4 Building Blocks

The SM scheme presented in this work is a modular construction and uses three components: continuous key-agreement (CKA), forward-secure authenticated encryption with associated data (FS-AEAD) and—for lack of a better name—PRF-PRNGs. These primitives are presented in isolation in this section before combining them into an SM scheme in Section 5.

4.1 Continuous Key Agreement

This work distills out the public-ratchet part of the Signal protocol and casts it as a separate primitive called *continuous key agreement (CKA).* This step is not only useful to improve the intuitive understanding of the various components of the Signal protocol and their interdependence, but it also increases modularity, which, for example, would—once the need arises—allow to replace the current CKA mechanism based on DDH by one that is post-quantum secure.

4.1.1 Defining CKA

At a high level, CKA is a synchronous two-party protocol between A and B. Odd rounds i consist of A sending and B receiving a message T_i, whereas in even rounds, B is the sender and A the receiver. Each round i also produces a key I_i, which is output by the sender upon sending T_i and by the receiver upon receiving T_i.

Definition 12. *A* continuous-key-agreement (CKA) scheme *is a quadruple of algorithms* $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}A}, \mathsf{CKA\text{-}Init\text{-}B}, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$*, where*

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

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

Correctness. A CKA scheme is correct if in the security game in Figure 3 (explained below), A and B always, i.e., with probability 1, output the same key in every round.

Security. The basic property a CKA scheme must satisfy is that conditioned on the transcript $T_1, T_2, \ldots$, the keys $I_1, I_2, \ldots$ look uniformly random and independent. An attacker against a CKA scheme is required to be passive, i.e., may not modify the messages T_i. However, it is given the power to possibly (1) control the random coins used by the sender and (2) leak the current state of either party. Correspondingly, the keys I_i produced under such circumstances need not be secure. The parties are required to recover from a state compromise *within* 2 *rounds.*[7]

The formal security game for CKA is provided in Figure 3. It begins with a call to the **init** oracle, which samples a bit b, initializes the states of both parties, and defines epoch counters t_{A} and t_{B}. Procedure **init** takes a value t^*, which determines in which round the challenge oracle may be called.

Upon completion of the initialization procedure, the attacker gets to interact arbitrarily with the remaining oracles, as long as *the calls are in a "ping-pong" order*, i.e., a call to a send oracle for A is followed by a receive call for B, then by a send oracle for B, etc. The attacker only gets to use the challenge oracle for epoch t^*. No corruption or using bad randomness (**send-A'** and **send-B'**) is allowed less than two epochs before the challenge is sent ($\mathsf{allow\text{-}corr}$).

[7]Of course, one could also parametrize the number of rounds required to recover (all CKA schemes in this work recover within two rounds, however).

Security Game for CKA

```
init $(t^*)$
    $k \leftarrow\$ \mathcal{K}$
    $\gamma^{\mathsf{A}} \leftarrow \mathsf{CKA\text{-}Init\text{-}A}(k)$
    $\gamma^{\mathsf{B}} \leftarrow \mathsf{CKA\text{-}Init\text{-}B}(k)$
    $t_{\mathsf{A}}, t_{\mathsf{B}} \leftarrow 0$
    $b \leftarrow\$ \{0,1\}$

corr-A
    req $\mathsf{allow\text{-}corr}$ or $\mathsf{finished}_{\mathsf{A}}$
    return $\gamma^{\mathsf{A}}$

send-A
    $t_{\mathsf{A}}$ ++
    $(\gamma, T_{t_{\mathsf{A}}}, I_{t_{\mathsf{A}}}) \leftarrow\$ \mathsf{CKA\text{-}S}(\gamma)$
    return $(T_{t_{\mathsf{A}}}, I_{t_{\mathsf{A}}})$

send-A' $(r)$
    $t_{\mathsf{A}}$ ++
    req $\mathsf{allow\text{-}corr}$
    $(\gamma, T_{t_{\mathsf{A}}}, I_{t_{\mathsf{A}}}) \leftarrow \mathsf{CKA\text{-}S}(\gamma; r)$
    return $(T_{t_{\mathsf{A}}}, I_{t_{\mathsf{A}}})$

receive-A
    $t_{\mathsf{A}}$ ++
    $(\gamma^{\mathsf{A}}, *) \leftarrow \mathsf{CKA\text{-}R}(\gamma^{\mathsf{A}}, T_{t_{\mathsf{A}}})$

chall-A
    $t_{\mathsf{A}}$ ++
    req $t_{\mathsf{A}} = t^*$
    $(\gamma, T_{t_{\mathsf{A}}}, I_{t_{\mathsf{A}}}) \leftarrow\$ \mathsf{CKA\text{-}S}(\gamma)$
    if $b = 0$
        return $(T_{t_{\mathsf{A}}}, I_{t_{\mathsf{A}}})$
    else
        $I \leftarrow\$ \mathcal{I}$
        return $(T_{t_{\mathsf{A}}}, I)$
```

$$\mathsf{allow\text{-}corr}_{\mathsf{P}} \;:\Longleftrightarrow\; \max(t_{\mathsf{A}}, t_{\mathsf{B}}) \le t^* - 2$$

$$\mathsf{finished}_{\mathsf{P}} \;:\Longleftrightarrow\; t_{\mathsf{P}} \ge t^* + \Delta_{\mathsf{CKA}}$$

Figure 3: *Oracles corresponding to party* A *of the CKA security game for a scheme* $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}A}, \mathsf{CKA\text{-}Init\text{-}B}, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$*; the oracles for* B *are defined analogously.*

The game is parametrized by Δ_{CKA}, which stands for the number of epochs that need to pass after t^* until the states do not contain secret information pertaining to the challenge. Once a party reaches epoch $t^* + \Delta_{\mathsf{CKA}}$, its state may be revealed to the attacker (via the corresponding corruption oracle). The game ends (not made explicit) once both states are revealed after the challenge phase. The attacker wins the game if it eventually outputs a bit $b' = b$.

The advantage of an attacker $\mathcal{A}$ against a CKA scheme CKA with $\Delta_{\mathsf{CKA}} = \Delta$ is denoted by $\mathrm{Adv}^{\mathsf{CKA}}_{\mathrm{ror},\Delta}(\mathcal{A})$. The attacker is parameterized by its running time t.

Definition 13. *A CKA scheme* CKA *is* (t, Δ, ε)-secure *if for all* t*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{CKA}}_{\mathrm{ror},\Delta}(\mathcal{A}) \;\le\; \varepsilon\ .$$

4.1.2 Instantiating CKA

This paper presents several instantiations of CKA: First, a generic CKA scheme with $\Delta = 0$ based on any key-encapsulation mechanism (KEM). Then, by considering the ElGamal KEM (cf. Section C.1) and observing that an encapsulated key can be “reused” as public key, one obtains a CKA scheme based on the decisional Diffie-Hellman (DDH) assumption, where the scheme saves a factor of 2 in communication compared to a straight-forward instantiation of the generic scheme. However, the scheme has $\Delta = 1$ (cf. Figure 4). A similar idea can also be applied to the Frodo KEM [6] (cf. Section C.2), which is based on learning with errors (LWE). A general study of KEMs for which key generation and encapsulation can be merged has recently been put forth in follow-up work by Drucker and Gueron [10].

CKA from KEMs. A CKA scheme with $\Delta = 0$ can be built from a KEM in natural way: in every epoch, one party sends a public key pk of a freshly generated key pair and an encapsulated key under the key pk' received from the other party in the previous epoch. Specifically, consider a CKA scheme $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}A}, \mathsf{CKA\text{-}Init\text{-}B}, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$ that is obtained from a KEM KEM as follows:

- The initial shared state $k = (\mathsf{pk}, \mathsf{sk})$ consists of a (freshly generated) KEM key pair. The initialization for A outputs $\mathsf{pk} \leftarrow \mathsf{CKA\text{-}Init\text{-}A}(k)$ and that for B outputs $\mathsf{sk} \leftarrow \mathsf{CKA\text{-}Init\text{-}B}(k)$.
- The send algorithm $\mathsf{CKA\text{-}S}$ takes as input the current state $\gamma = \mathsf{pk}$ and proceeds as follows: It
  1. encapsulates a key $(c, I) \leftarrow\$ \mathsf{Enc}(\mathsf{pk})$,
  2. generates a new key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{Gen}$,
  3. sets the CKA message to $T \leftarrow (c, \mathsf{pk})$,
  4. sets the new state to $\gamma \leftarrow \mathsf{sk}$, and
  5. returns (γ, T, I).
- The receive algorithm $\mathsf{CKA\text{-}R}$ takes as input the current state $\gamma = \mathsf{sk}$ as well as a message $T = (c, \mathsf{pk})$ and proceeds as follows: It
  1. decapsulates the key $I \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$,
  2. sets the new state to $\gamma \leftarrow \mathsf{pk}$, and
  3. returns (γ, I).

Theorem 2. *Assume* KEM *is a* (t', ε)*-secure KEM. Then, the above CKA scheme* CKA *is* (t, Δ, ε)*-secure for* $t \approx t'$ *and* $\Delta = 0$.

Proof. The theorem is proved by showing that for every attacker $\mathcal{A}$ against the CKA security of CKA, there exists an attacker $\mathcal{B}$ such that

$$\mathrm{Adv}^{\mathsf{CKA}}_{\mathrm{ror},\Delta}(\mathcal{A}) \;\leq\; \mathrm{Adv}^{\mathsf{KEM}}_{\mathrm{cpa}}(\mathcal{B}) \; .$$

Assume w.l.o.g. that t^* is *odd*, i.e., A sends the challenge; the case where t^* is even is handled analogously. The reduction $\mathcal{B}$ is straight-forward: Let (pk, c, k) be a KEM challenge (where k is either embedded in c or random and independent). Then,

- in epoch $t^* - 1$, the reduction embeds pk into B's CKA message T_{t^*-1}, and
- in epoch t^*, the reduction embeds c into A's message T_{t^*} and uses k as the CKA key I_{t^*} that A outputs.

Observe that A's simulated state may safely be revealed once she reaches epoch $t^* + \Delta_{\mathsf{CKA}} = t^*$ since at that ponit her state contains a secret key that is unrelated to the challenge. Similarly, once B reaches t^*, he deletes the secret key used to decapsulate, and thus the reduction need not know it to simulate B's state at the end. $\square$

CKA from DDH. Observe that if one instantiates the above KEM-based CKA scheme with the ElGamal KEM (cf. Section C.1)ver some group G, both the public key and the encapsulated key are elements of G. Hence, the Signal protocol uses an optimization of the ElGamal KEM where a single group element first serves as an encapsulated key sent by, say, A and then as the public key B uses to encapsulate his next key. Interestingly, this comes at the price of having $\Delta = 1$ (as opposed to $\Delta = 0$) due to the need for parties to hold on to their exponents (which serve both as secret keys and encapsulation randomness) longer.

Concretely, a CKA scheme $\mathsf{CKA} = (\mathsf{CKA\text{-}Init\text{-}A}, \mathsf{CKA\text{-}Init\text{-}B}, \mathsf{CKA\text{-}S}, \mathsf{CKA\text{-}R})$ can be obtained from the DDH assumption[8] in a cyclic group $G = \langle g \rangle$ as follows:

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

For the theorem below, let a group G be (t, ε)-secure if every attacker with running time at most t has advantage at most ε at distinguishing DDH triples from random triples.

Theorem 3. *Assume group G is (t, ε)-DDH-secure. Then, the above CKA scheme* CKA *is (t, Δ, ε)-secure for $t \approx t'$ and $\Delta = 1$.*

Proof. Assume w.l.o.g. that t^* is *odd*, i.e., A sends the challenge; the case where t^* is even is handled analogously. Let (g^a, g^b, g^c) be a Diffie-Hellman or random triple. The reduction simulates the CKA protocol in the straight-forward way but embeds the triple into the CKA as follows:

- in epoch $t^* - 1$, it uses $T_{t^*-1} = g^a$ and $I_{t^*-1} = g^{xa}$, where x is the exponent used to simulate $T_{t^*-2} = g^x$.
- in epoch t^*, it uses $T_{t^*} = g^b$ and $I_{t^*} = g^c$; and
- in epoch $t^* + 1$, for a random x', it uses $T_{t^*+1} = g^{x'}$ and $I_{t^*+1} = g^{bx'}$.

[8]The DDH assumption states that it is hard to distinguish DH triples (g^a, g^b, g^{ab}) from random triples (g^a, g^b, g^c), where a, b, and c are uniformly random and independent exponents.

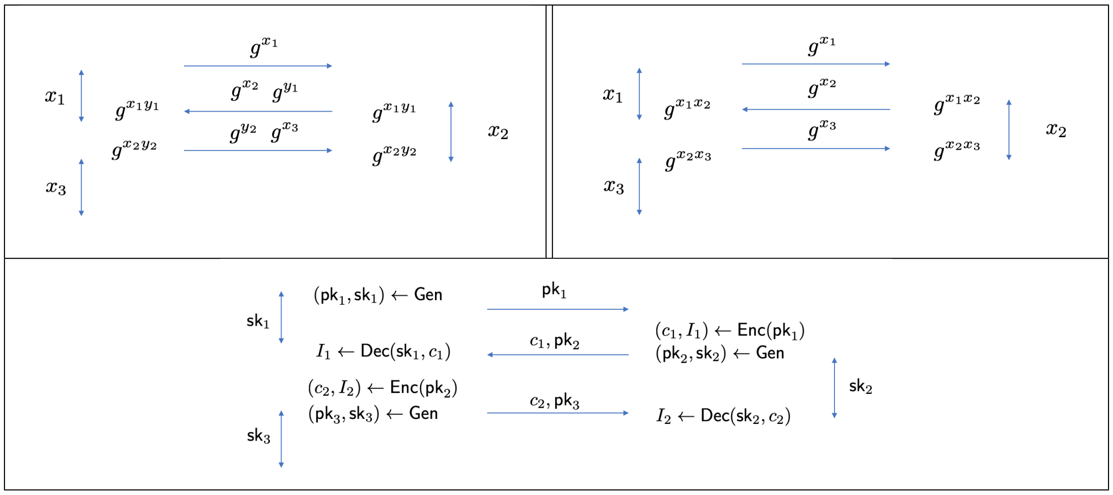
g^{x_1}
x_1
$g^{x_1y_1}$
g^{x_2} g^{y_1}
$g^{x_1y_1}$
x_2
$g^{x_2y_2}$
g^{y_2} g^{x_3}
$g^{x_2y_2}$
x_3
g^{x_1}
x_1
$g^{x_1x_2}$
g^{x_2}
$g^{x_1x_2}$
x_2
g^{x_3}
$g^{x_2x_3}$
$g^{x_2x_3}$
x_3
$(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}$
pk_1
sk_1
$(c_1, I_1) \leftarrow \mathsf{Enc}(\mathsf{pk}_1)$
$I_1 \leftarrow \mathsf{Dec}(\mathsf{sk}_1, c_1)$
c_1, pk_2
$(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}$
$(c_2, I_2) \leftarrow \mathsf{Enc}(\mathsf{pk}_2)$
sk_2
$(\mathsf{pk}_3, \mathsf{sk}_3) \leftarrow \mathsf{Gen}$
c_2, pk_3
$I_2 \leftarrow \mathsf{Dec}(\mathsf{sk}_2, c_2)$
sk_3

Figure 4: *Comparison between CKAs with $\Delta_{\mathsf{CKA}} = 0$ based on generic KEMs (bottom) resp. El-Gamal (left) and ElGamal-based CKA with $\Delta_{\mathsf{CKA}} = 1$ (right). The figure shows the lifetimes of secret state values alongside the protocol. Consider the state of parties A and B once the third protocol message is delivered: In the $\Delta_{\mathsf{CKA}} = 0$ protocols, A has a secret key sk_3 resp. an exponent x_3 that is not used yet, and B's state contains no secret information; hence, no past keys can be computed upon state compromise. In the ElGamal protocol with $\Delta_{\mathsf{CKA}} = 1$, A stores secret exponent x_3, which allows to compute key $g^{x_2x_3}$.*

It is easy to verify that this correctly simulates the CKA experiment. In particular, note that A's simulated state may safely be revealed once she reaches epoch $t^* + \Delta_{\mathsf{CKA}} = t^* + 1$ since at that point she would have deleted her exponent b in the actual game. B would delete his exponent a as soon as he reaches t^*, i.e., even before $t^* + \Delta_{\mathsf{CKA}}$. □

CKA based on LWE. A similar public-key-ciphertext-reuse idea can be applied to the Frodo KEM [6] (cf. Section C.2 for an explanation of notation), which is based on learning with errors (LWE). Concretely, a CKA scheme, in which A and B actually have slightly different algorithms for sending and receiving, can be obtained as shown next. The scheme is parametrized by a size parameters n and $\bar{n}$, a modulus q, and an error distribution χ over $\mathbb{Z}_q$.

- The initial shared state k consists of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ chosen uniformly at random, and independently sampled matrices $\mathbf{S}, \mathbf{E} \leftarrow \chi(\mathbb{Z}_q^{n\times\bar{n}})$, where $\chi(\mathbb{Z}_q^{n\times\bar{n}})$ is the distribution on matrices obtained by sampling each entry independently according to χ. A initially stores $(\mathbf{A}, \mathbf{S}) \leftarrow \mathsf{CKA\text{-}Init\text{-}A}(k)$, and B's initial state is $(\mathbf{A}, \mathbf{B} := \mathbf{AS} + \mathbf{E}) \leftarrow \mathsf{CKA\text{-}Init\text{-}B}(k)$.
- B's send algorithm $\mathsf{CKA\text{-}S\text{-}B}$ takes as input the current state $\gamma = (\mathbf{A}, \mathbf{B})$ and proceeds as follows: It
  1. chooses $\mathbf{S}', \mathbf{E}' \sim \chi(\mathbb{Z}_q^{\bar{n}\times n})$, and $\tilde{\mathbf{E}}' \sim \chi(\mathbb{Z}_q^{\bar{n}\times\bar{n}})$ independently,
  2. computes $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$,
  3. computes $\mathbf{V}' \leftarrow \mathbf{S}'\mathbf{B} + \tilde{\mathbf{E}}'$,
  4. computes reconciliation information $\mathbf{C}' \leftarrow \langle \mathbf{V}' \rangle_{2^B}$,
  5. computes the key $I \leftarrow \lfloor \mathbf{V}' \rceil_{2^B}$,

6. sets the CKA message to $c = (\mathbf{B}', \mathbf{C}')$,
7. sets the new state to $\gamma \leftarrow (\mathbf{A}, \mathbf{S}')$, and
8. returns (γ, T, I).

- A's send algorithm CKA-S-A proceeds analogously, but it multiplies by $\mathbf{S}'$ from the right.
- A's receive algorithm CKA-R-A takes as input the current state $\gamma = (\mathbf{A}, \mathbf{S})$ as well as a message $T = (\mathbf{B}', \mathbf{C}')$ and proceeds as follows: It

  1. computes the key $I = \mathsf{rec}(\mathbf{B}'\mathbf{S}, \mathbf{C}')$,
  2. sets the new state to $\gamma \leftarrow (\mathbf{A}, \mathbf{B}')$, and
  3. returns (γ, I).

- B's receive algorithm CKA-R-B proceeds analogously, but it multiplies by $\mathbf{S}'$ from the left.

The security of the above CKA scheme based on LWE is proved by reduction to the CPA security of the Frodo KEM.

Theorem 4. *Assume Frodo is a (t, ε)-CPA-secure KEM. Then, the above CKA scheme* CKA *is (t, Δ, ε)-secure for $t \approx t'$ and $\Delta = 1$.*

Proof. Depending on the challenge epoch t^*, the reduction is either to the security of the original Frodo (as laid out in Section C.2) or to a variant where the encryptor multiplies from the right and the decryptor from the left. For this proof sketch, assume w.l.o.g. that t^* is even, i.e., the challenge ciphertext is sent by B and the reduction is to the original Frodo scheme.

Similarly to all previous CKA proofs, the reduction simply embeds a particular instance of the Frodo KEM into the full CKA game. Specifically, let $\mathbf{A}$, $\mathbf{B}$, $\mathbf{V}'$, $\mathbf{C}'$, and K be the KEM instance, where K is either the key as computed by Frodo or uniformly random.

In epoch $t^* - 1$, the reduction embeds $\mathbf{B}$ into the execution of CKA-S-A, instead of choosing values $\mathbf{S}$ and $\mathbf{E}$ and computing $\mathbf{B} \leftarrow \mathbf{A}\mathbf{S} + \mathbf{E}$ itself. It then picks $\tilde{\mathbf{E}} \sim \chi(\mathbb{Z}_q^{n \times \bar{n}})$, sets $\mathbf{V} \leftarrow \mathbf{B}'_{\mathsf{t}^*-2}\mathbf{S} + \tilde{\mathbf{E}}$ (where $\mathbf{B}'_{\mathsf{t}^*-2}$ was sent by B in epoch $t^* - 2$), and computes $\mathbf{C} = \langle \mathbf{V} \rangle_{2^B}$ as well as $I \leftarrow \lfloor \mathbf{V} \rceil_{2^B}$. Once B obtains $\mathbf{B}$ and $\mathbf{C}$, CKA-R-B recovers the key I as $\mathsf{rec}(\mathbf{S}'\mathbf{B}, \mathbf{C})$.

In epoch t^*, the reduction embeds $\mathbf{B}'$, $\mathbf{V}'$, $\mathbf{C}'$, and $I \leftarrow K$ into CKA-S-B, instead of computing these values as CKA-S-B normally would. Once A receives $\mathbf{B}'$ and $\mathbf{C}'$, CKA-R-A simply outputs K instead of computing $\mathsf{rec}(\mathbf{B}'\mathbf{S}, \mathbf{C}')$.

By inspection, one can verify that this correctly simulates the CKA experiment. In particular, note that B's simulated state may safely be revealed once he reaches epoch $t^* + \Delta_{\mathsf{CKA}} = t^* + 1$ since at that point he would have deleted his secret value $\mathbf{S}'$ in the actual game. A would delete her secret value $\mathbf{S}$ as soon as she reaches t^*, i.e., even before $t^* + \Delta_{\mathsf{CKA}}$. □

4.2 Forward-Secure AEAD

4.2.1 Defining FS-AEAD

Forward-secure authenticated encryption with associated data is a stateful primitive between a sender A and a receiver B and can be considered a single-epoch variant of an SM scheme, a fact that is also evident from its security definition, which resembles that of SM schemes.

Definition 14. Forward-secure authenticated encryption with associated data (FS-AEAD) *is a tuple of algorithms* $\mathsf{FS\text{-}AEAD} = (\mathsf{FS\text{-}Init\text{-}S}, \mathsf{FS\text{-}Init\text{-}R}, \mathsf{FS\text{-}Send}, \mathsf{FS\text{-}Rcv})$, *where*

- $\mathsf{FS\text{-}Init\text{-}S}$ *(and similarly* $\mathsf{FS\text{-}Init\text{-}R}$*) takes a key k and outputs a state* $v_{\mathsf{A}} \leftarrow \mathsf{FS\text{-}Init\text{-}S}(k)$*,*
- $\mathsf{FS\text{-}Send}$ *takes a state v, associated data a, and a message m and produces a new state and a ciphertext* $(v', e) \leftarrow \mathsf{FS\text{-}Send}(v, a, m)$*, and*
- $\mathsf{FS\text{-}Rcv}$ *takes a state v, associated data a, and a ciphertext e and produces a new state, an index, and a message* $(v', i, m) \leftarrow \mathsf{FS\text{-}Rcv}(v, a, e)$*.*

Observe that all algorithms of an FS-AEAD scheme are deterministic.

Memory management. In addition to the basic syntax above, it is useful to define the following two functions $\mathsf{FS\text{-}Stop}$ (called by the sender) and $\mathsf{FS\text{-}Max}$ (called by the receiver) for memory management:

- $\mathsf{FS\text{-}Stop}$, given an FS-AEAD state v, outputs how many messages have been received and then "erases" the FS-AEAD session corresponding to v form memory; and
- $\mathsf{FS\text{-}Max}$, given a state v and an integer ℓ, remembers ℓ internally such that the session corresponding to v is erased from memory as soon as ℓ messages have been received.

These features will be useful in the full protocol (cf. Section 5) to be able to terminate individual FS-AEAD sessions when they are no longer needed. Providing a formal requirement for these additional functions is omitted. Moreover, since an attacker can infer the value of the message counter from the behavior of the protocol anyway, there is no dedicated oracle included in the security game below.

Correctness and security. Both correctness and security are built into the security game depicted in Figure 5. The game is the single-epoch analogue of the SM security game (cf. Figure 2) and therefore has similarly defined oracles and similar record keeping. A crucial difference is that as soon as the receiver B is compromised, the game ends with a full state reveal as no more security can be provided. If only the sender A is compromised, the game continues and uncompromised messages must remain secure.

The advantage of an attacker $\mathcal{A}$ against an FS-AEAD scheme $\mathsf{FS\text{-}AEAD}$ is denoted by the expression $\mathrm{Adv}^{\mathsf{FS\text{-}AEAD}}_{\text{fs-aead}}(\mathcal{A})$. The attacker is parameterized by its running time t and the total number of queries q it makes.

Definition 15. *An FS-AEAD scheme* $\mathsf{FS\text{-}AEAD}$ *is* (t, q, ε)-secure *if for all* (t, q)*-attackers* $\mathcal{A}$*,*

$$\mathrm{Adv}^{\mathsf{FS\text{-}AEAD}}_{\text{fs-aead}}(\mathcal{A}) \ \leq \ \varepsilon \ .$$

4.2.2 Instantiating FS-AEAD

An FS-AEAD scheme can be easily constructed from two components:

- an AEAD scheme $\mathsf{AE} = (\mathsf{Enc}, \mathsf{Dec})$, and
- a PRG $G : \mathcal{W} \to \mathcal{W} \times \mathcal{K}$, where $\mathcal{K}$ is the key space of the AEAD scheme.

Security Game for FS-AEAD

```
init
    $k \leftarrow\$ \mathcal{K}$
    $v_A \leftarrow$ FS-Init-S$(k)$
    $v_B \leftarrow$ FS-Init-R$(k)$
    $i_A \leftarrow 0$
    corr$_A \leftarrow$ false
    trans, chall, comp $\leftarrow \emptyset$
    $b \leftarrow\$ \{0,1\}$

corr-A
    corr$_A \leftarrow$ true
    return $v_A$

corr-B
    req chall $= \emptyset$
    end $(v_A, v_B)$

transmit-A $(a, m)$
    $i_A$ ++
    $(v_A, e) \leftarrow$ FS-Send$(v_A, a, m)$
    record(good, $a, m, e$)
    return $e$

chall-A $(a, m_0, m_1)$
    req $\neg$corr$_A$ and $|m_0| = |m_1|$
    $i_A$ ++
    $(v_A, e) \leftarrow$ FS-Send$(v_A, a, m_b)$
    record(chall, $a, m_b, e$)
    return $e$

deliver-B $(a, e)$
    req $(i, a, m, e) \in$ trans
        for some $i, m$
    $(v_B, i', m') \leftarrow$ FS-Rcv$(v_B, a, e)$
    if $(i', m') \neq (i, m)$
        win
    if $(i, m) \in$ chall
        $m' \leftarrow \bot$
    delete$(i)$
    return $(i', m')$

inject-B $(a, e)$
    req $(a, e) \notin$ trans
    $(v_B, i', m') \leftarrow$ FS-Rcv$(v_B, a, e)$
    if $m' \neq \bot$ and $i' \notin$ comp
        win
    delete$(i')$
    return $(i', m')$
```

```
record (flag, $a, m, e$)
    rec $\leftarrow (i_A, a, m, e)$
    trans $\overset{+}{\leftarrow}$ rec
    if flag = bad or corr$_A$
        comp $\overset{+}{\leftarrow}$ rec
    if flag = chall
        chall $\overset{+}{\leftarrow}$ rec

delete $(i)$
    rec $\leftarrow (i, a, m, e)$ for $m, a, e$
            s.t. $(i, a, m, e) \in$ trans
    trans, chall, comp $\overset{-}{\leftarrow}$ rec
```

Figure 5: *Oracles corresponding to party* A *of the FS-AEAD security game for a scheme* FS-AEAD = (FS-Init-S, FS-Init-R, FS-Send, FS-Rcv).

The scheme is described in Figure 6. For simplicity the states of sender A and receiver B are is not made explicit; it consists of the variables set during initialization. The main idea of the scheme, is that the A and B share the state w of a PRG G. State w is initialized with a pre-shared key $k \in \mathcal{W}$, which is assumed to be chosen uniformly at random. Both parties keep local counters i_A and i_B, respectively.[9] A, when sending the i^{th} message m with associated data (AD) a, uses G to expand the current state to a new state and an AEAD key $(w, K) \leftarrow G(w)$ and computes an AEAD encryption under K of m with AD $h = (i, a)$.

Since B may receive ciphertext out of order, whenever he receives a ciphertext, he first checks whether the key is already stored in a dictionary $\mathcal{D}$. If the index of the message is higher than expected (i.e., larger than $i_B + 1$), B skips the PRG ahead and stores the skipped keys in $\mathcal{D}$. In either case, once the key is obtained, it is used to decrypt. If decryption fails, FS-Rcv throws an exception (**error**), which causes the state to be rolled back to where it was before the call to FS-Rcv.

[9]For ease of description, the FS-AEAD state of the parties is not made explicit as a variable v.

Forward-Secure AEAD

```
Init-A ($k$)
    $w \leftarrow k$
    $i_{\mathsf{A}} \leftarrow 0$

Init-B ($k$)
    $w \leftarrow k$
    $i_{\mathsf{B}} \leftarrow 0$
    $\mathcal{D}[\cdot] \leftarrow \lambda$

try-skipped ($i$)
    $K \leftarrow \mathcal{D}[i]$
    $\mathcal{D}[i] \leftarrow \bot$
    return $K$

FS-Send ($a, m$)
    $i_{\mathsf{A}}$ ++
    $(w, K) \leftarrow G(w)$
    $h \leftarrow (i_{\mathsf{A}}, a)$
    $e \leftarrow \mathsf{Enc}(K, h, m)$
    return $(i_{\mathsf{A}}, e)$

skip ($u$)
    while $i_{\mathsf{B}} < u - 1$
        $i_{\mathsf{B}}$ ++
        $(w, K) \leftarrow G(w)$
        $\mathcal{D}[u] \leftarrow K$

FS-Rcv ($a, c$)
    $(i, e) \leftarrow c$
    $K \leftarrow$ try-skipped($i$)
    if $K = \bot$
        skip($i$)
        $(w, K) \leftarrow G(w)$
        $i_{\mathsf{B}} \leftarrow i$
    $h \leftarrow (i, a)$
    $m \leftarrow \mathsf{Dec}(K, h, e)$
    if $m = \bot$
        error
    return $(i, m)$
```

Figure 6: *FS-AEAD scheme based on AEAD and a PRG.*

Theorem 5. *Assume* AE *is a* $(t', \varepsilon_{\text{aead}})$*-OT-CCA secure AEAD scheme and* G *is a* $(t', \varepsilon_{\text{prg}})$*-secure PRG. Then, the above FS-AEAD scheme* $\mathsf{FS\text{-}AEAD}$ *is* (t, q, ε)*-secure for* $t \approx t'$ *and*

$$\varepsilon \;\leq\; q \cdot (\varepsilon_{\text{aead}} + \varepsilon_{\text{prg}}) \; .$$

Proof. The proof is a straight-forward hybrid argument: Let H_0 denote the actual FS-AEAD security game.

- In the first hybrid experiment H_1, all PRG outputs are replaced by uniform random values. The indistinguishability of H_1 from the original experiment H_0 follows immediately form the security of the PRG.
- In the second hybrid experiment H_2, the **win** condition inside the **deliver-A** oracle is removed. The prefect correctness of the proposed FS-AEAD scheme is easily seen by inspection, and hence H_1 and H_2 are indistinguishable.
- In the third hybrid experiment H_3, all AEAD ciphertexts are replaced by random ciphertexts and any (uncompromised) injections are always rejected. Since in H_2 all keys used for the AEAD scheme are random, the indistinguishability of H_3 from H_2 follows immediately from the security of the AEAD scheme. Moreover, the advantage of any attacker in H_3 is 0.

□

4.3 PRF-PRNGs

4.3.1 Defining PRF-PRNGs

A *PRF-PRNG* resembles both a pseudo-random function (PRF) and a pseudorandom number generator with input (PRNG)—hence the name. On the one hand, as a PRNG would, a PRF-PRNG (1) repeatedly accepts inputs I and uses them to refresh its state σ and (2) occasionally

Security Game for PRF-PRNG

```
init
    $k \leftarrow\$ \mathcal{K}$
    $\sigma \leftarrow \mathsf{P\text{-}Init}(k)$
    $\mathsf{corr} \leftarrow \mathsf{false}$
    $\mathsf{prng}, \mathsf{prf} \leftarrow \mathsf{false}$
    sample random function $\mathcal{F}$
    $b \leftarrow\$ \{0,1\}$

corr
    req $\neg\mathsf{prf}$
    return $\sigma$

process $(I)$
    $I \leftarrow$ sam-if-nec$(I)$
    $(\sigma, R) \leftarrow \mathsf{P\text{-}Up}(\sigma, I)$
    return $R$

chall-prf $(I)$
    req $\neg\mathsf{corr}$ and $\neg\mathsf{prng}$
    $\mathsf{prf} \leftarrow \mathsf{true}$
    $(\sigma', R) \leftarrow \mathsf{P\text{-}Up}(\sigma, I)$
    if $b = 1$
        $R \leftarrow \mathcal{F}(I)$
    return $(\sigma', R)$

chall-prng $(I)$
    $I \leftarrow$ sam-if-nec$(I)$
    req $\neg\mathsf{corr}$ and $\neg\mathsf{prf}$
    $\mathsf{prng} \leftarrow \mathsf{true}$
    $(\sigma, R) \leftarrow \mathsf{P\text{-}Up}(\sigma, I)$
    if $b = 1$
        $R \leftarrow\$ \mathcal{R}$
    return $R$
```

```
sam-if-nec $(I)$
    if $I = \perp$
        $I \leftarrow\$ \mathcal{I}$
        $\mathsf{corr} \leftarrow \mathsf{false}$
    return $I$
```

Figure 7: *Oracles of the PRF-PRNG security game for a scheme* $\mathsf{P} = (\mathsf{P\text{-}Init}, \mathsf{P\text{-}Up})$.

uses the state, provided it has sufficient entropy, to derive a pseudo-random pair of output R and new state; for the purposes of secure messaging, it suffices to combine properties (1) and (2) into a single procedure. On the other hand, a PRF-PRNG can be used as a PRF in the sense that if the state has high entropy, the answers to various inputs I *on the same state* are indistinguishable from random and independent values.

Definition 16. *A* PRF-PRNG *is a pair of algorithms* $\mathsf{P} = (\mathsf{P\text{-}Init}, \mathsf{P\text{-}Up})$*, where*

- $\mathsf{P\text{-}Init}$ *takes a key* k *and produces a state* $\sigma \leftarrow \mathsf{P\text{-}Init}(k)$*, and*
- $\mathsf{P\text{-}Up}$ *takes a state* σ *and an input* I *and produces a new state and an output* $(\sigma', R) \leftarrow \mathsf{P\text{-}Up}(\sigma, I)$*.*

Security. The simple intuitive security requirement for a double-seed PRG is that $\mathsf{P\text{-}Init}(\sigma, I)$ produce a pseudorandom value if the state σ is uncorrupted (i.e., has high entropy) or the input I is chosen uniformly from some set $\mathcal{S}$. Moreover, if the state is uncorrupted, it should have the PRF property described above. This is captured by the security definition described by Figure 7:

- **Initialization:** Procedure **init** chooses a random bit b, initializes the PRF-PRNG with a random key, and sets two flags prng and prf to false: the PRNG and PRF modes are mutually exclusive and only one type of challenge may be called; the flags keep track of which. During initialization, a random function $\mathcal{F}$ with appropriate input and output lengths is chosen to create PRF challenges (see below).

- **PRNG mode:** The oracle **process** can be called in two ways: either I is an input specified by the attacker and is simply absorbed into the state, or $I = \perp$, in which case the game chooses it randomly (inside **sam-if-nec**) and absorbs it into the state, which at this point becomes uncorrupted. Oracle **chall-prng** is works in the same fashion but creates a challenge.
- **PRF mode:** Once the state is uncompromised the attacker can decide to obtain PRF challenges by calling **chall-prf**, which simply evaluates the (adversarially chosen) input on the current state without updating it and creates a challenge (using random function $\mathcal{F}$ when $b = 1$).
- **Corruption:** At any time, except after asking for PRF challenges, the attacker may obtain the state by calling **corr**.

The advantage of $\mathcal{A}$ in the PRF-PRNG game is denoted by $\mathrm{Adv}^{\mathsf{P}}_{\mathrm{PP}}(\mathcal{A})$. The attacker is parameterized by its running time t.

Definition 17. *An PRF-PRG* P *is* (t, ε)-secure *if for all t-attackers $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathsf{P}}_{\mathrm{PP}}(\mathcal{A}) \;\leq\; \varepsilon \; .$$

4.3.2 Instantiating PRF-PRNGs

Being a PRF-PRNG is a property the HKDF function used by Signal is assumed to have; in particular, Marlinspike and Perrin [25] recommend the primitive be implemented with using HKDF [21] with SHA-256 or SHA-512 [27] where the state σ is used as HKDF salt and I as HKDF input key material. This paper therefore merely reduces the security of the presented schemes to the PRF-PRNG security of whatever function is used to instantiate it.

Alternatively, a simple standard-model instantiation (whose rather immediate proof is omitted) can be based on a pseudorandom *permutation* (PRP) $\Pi : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and a PRG $G : \{0,1\}^n \to \{0,1\}^n \times \mathcal{K}$ by letting the state be the PRP key $s \in \{0,1\}^n$ and

$$(s', R) \leftarrow \mathsf{P\text{-}Up}(s, I) \;=\; G(\Pi_s(I)) \; .$$

5 Secure Messaging Scheme

This section presents a Signal-based secure messaging (SM) scheme and establishes its security under Definition 8. The scheme suitably and modularly combines continuous key-agreement (CKA), forward-secure authenticated encryption with associated data (FS-AEAD), and PRF-PRNGs; these primitives are explained in detail in Section 4.

5.1 The Scheme

The scheme is inspired by the Signal protocol, but differs from it in a few points, as explained in Section 5.2. The main idea of the scheme is that the parties A and B keep track of the same PRF-PRG (aka the "root RNG"), which they use to generate keys for FS-AEAD instances as needed. The root RNG is continuously refreshed by random values output by a CKA scheme that is run "in parallel."

Signal-Based Secure-Messaging Scheme

```
Init-A (k)
    id ← A
    $(k_{root}, k_{CKA}) \leftarrow k$
    $\sigma_{root} \leftarrow$ P-Init$(k_{root})$
    $(\sigma_{root}, k) \leftarrow$ P-Up$(\sigma_{root}, \lambda)$
    $v[\cdot] \leftarrow \lambda$
    $v[0] \leftarrow$ FS-Init-R$(k)$
    $\gamma \leftarrow$ CKA-Init-A$(k_{CKA})$
    $T_{cur} \leftarrow \lambda$
    $\ell_{prv} \leftarrow 0$
    $t_{cur} \leftarrow 0$

Send-A (m)
    if $t_{cur}$ is even
        $\ell_{prv} \leftarrow$ FS-Stop$(v[t_{cur} - 1])$
        $t_{cur}$ ++
        $(\gamma, T_{cur}, I) \leftarrow_{\$}$ CKA-S$(\gamma)$
        $(\sigma_{root}, k) \leftarrow$ P-Up$(\sigma_{root}, I)$
        $v[t_{cur}] \leftarrow$ FS-Init-S$(k)$
    $h \leftarrow (t_{cur}, T_{cur}, \ell_{prv})$
    $(v[t_{cur}], e) \leftarrow$ FS-Send$(v[t_{cur}], h, m)$

    return $(h, e)$

Rcv-A (c)
    $(h, e) \leftarrow c$
    $(t, T, \ell) \leftarrow h$
    req $t$ even and $t \le t_{cur} + 1$
    if $t = t_{cur} + 1$
        $t_{cur}$ ++
        FS-Max$(v[t-2], \ell)$
        $(\gamma, I) \leftarrow$ CKA-R$(\gamma, T)$
        $(\sigma_{root}, k) \leftarrow$ P-Up$(\sigma_{root}, I)$
        $v[t] \leftarrow$ FS-Init-R$(k)$
    $(v[t], i, m) \leftarrow$ FS-Rcv$(v[t], h, e)$
    if $m = \perp$
        error
    return $(t, i, m)$
```

Figure 8: *Secure-messaging scheme based on a FS-AEAD, a CKA scheme, and a PRF-PRNG. The figure only shows the algorithms for* A*;* B*'s algorithms are analogous, with "even" replaced by "odd" (and with* Init-B *initializing* $v[0]$ *via* FS-Init-S*).*

State. Scheme SM keeps an internal state s_A (resp. s_B), which is initialized by Init-A (resp. Init-B) and used as well as updated by Send and Rcv. The state s_A of SM consists of the following values:

- an ID field with id = A,
- the state σ_{root} of the root RNG,
- states $v[0], v[1], v[2], \ldots$ of the various FS-AEAD instances,
- the state γ of the CKA scheme,
- the current CKA message T_{cur}, and
- an epoch counter t_{cur}.

In order to remove expired FS-AEAD sessions from memory, there is also a variable ℓ_{prv} that remembers the number of messages sent in the second most recent epoch. Recall (cf. Section 4.2) that once the maximum number of messages has been set via FS-Max, a session "erases" itself from the memory, and similarly for calling FS-Stop on a particular FS-AEAD session. For simplicity, removing the corresponding $v[t]$ from memory is not made explicit in either case. The state s_B is defined analogously.

The algorithms. The algorithms of scheme SM are depicted in Figure 8 and described in more detail below. For ease of description, the algorithms Send and Rcv are presented as Send-A and Rcv-A, which handle the case where id = A; the case id = B works analogously. Moreover, to improve readability, the state s_A is not made explicit in the description: it consists of the variables set by the initialization algorithm.

- **Initialization:** The initialization procedure $\mathsf{Init\text{-}A}$ expects a key k shared between A and B; k is assumed to have been created at some point before the execution during a trusted setup phase and to consist of initialization keys k_{root} and k_{CKA} for the root RNG and the CKA scheme, respectively. In a second step, the root RNG is initialized with k. Then, it is used to generate a key for FS-AEAD epoch $v[0]$; A acts as receiver in $v[0]$ and all subsequent even epochs and as sender in all subsequent odd epochs. Furthermore, $\mathsf{Init\text{-}A}$ also initializes the CKA scheme and sets the initial epoch $t_{\mathsf{cur}} \leftarrow 0$ and T_{cur} to a default value.[10]

As pointed out above, scheme SM runs a CKA protocol in parallel to sending its messages. To that end, A's first message includes the first message T_1 output by $\mathsf{CKA\text{-}S}$. All subsequent messages sent by A include T_1 until some message received from B includes T_2. At that point A would run $\mathsf{CKA\text{-}S}$ again and include T_3 with all her messages, and so on (cf. Section 4.1).

Upon either sending or receiving T_i for odd or even i, respectively, the CKA protocol also produces a random value I_i, which A absorbs into the root RNG. The resulting output k is used as key for a new FS-AEAD epoch.

- **Sending messages:** Procedure $\mathsf{Send\text{-}A}$ allows A to send a message to B. As a first step, $\mathsf{Send\text{-}A}$ determines whether it is A's turn to send the next CKA message, which is the case if t_{cur} is even. Whenever it is A's turn, $\mathsf{Send\text{-}A}$ runs $\mathsf{CKA\text{-}S}$ to produce the her next CKA message T and key I, which is absorbed into the root RNG. The resulting value k is used as a the key for a new FS-AEAD epoch, in which A acts as sender. The now old epoch is terminated by calling $\mathsf{FS\text{-}Stop}$ and the number of messages in the old epoch is stored in ℓ_{prv}, which will be sent along inside the header for every message of the new epoch.

  Irrespective of whether it was necessary to generate a new CKA message and generate a new FS-AEAD epoch, $\mathsf{Send\text{-}A}$ creates a header $h = (t_{\mathsf{cur}}, T_{\mathsf{cur}}, \ell_{\mathsf{prv}})$, and uses the current epoch $v[t_{\mathsf{cur}}]$ to get a ciphertext for (h, m) (where h is treated as associated data).

- **Receiving messages:** When a ciphertext $c = (h, e, \ell)$ with header $h = (t, T, \ell)$ is processed by $\mathsf{Rcv\text{-}A}$, there are two possibilities:[11]

  - $t \le t_{\mathsf{cur}}$: In this case, ciphertext c pertains to an existing FS-AEAD epoch, in which case $\mathsf{FS\text{-}Send}$ is simply called on $v[t]$ to process e. If the maximum number of messages has been received for session $v[t]$, the session is removed from memory.
  - $t = t_{\mathsf{cur}} + 1$ (in which case t_{cur} is odd): Here, the receiver algorithm advances t_{cur} by incrementing it and processes T with $\mathsf{CKA\text{-}R}$. This produces a key I, which is absorbed into the PRF-PRG to obtain a key k with which to initialize a new epoch $v[t_{\mathsf{cur}}]$ as receiver. Then, e is processed by $\mathsf{FS\text{-}Rcv}$ on $v[t_{\mathsf{cur}}]$. Note that Rcv also uses $\mathsf{FS\text{-}Max}$ to store ℓ as the maximum number of messages in the previous receive epoch.

  Irrespective of whether a new CKA message was received and a new epoch created, if e is rejected by $\mathsf{FS\text{-}Rcv}$, the algorithm raises an exception (**error**), which causes the entire state s_{A} to be rolled back to what it was before $\mathsf{Rcv\text{-}A}$ was called.

[10] B also starts in epoch $t_{\mathsf{cur}} \leftarrow 0$.

[11] Observe that c is only accepted if t_{cur} is even.

5.2 Differences to Signal

By instantiating the building blocks as shown below, one obtains an SM scheme that is very close to the actual Signal protocol (cf. [25, Section 5.2] for more details):

- **CKA:** the DDH-based CKA scheme from Section 4.1.2 using Curve25519 or Curve448 as specified in [22];
- **FS-AEAD:** FS-AEAD scheme from Section 4.2 with HMAC [20] with SHA-256 or SHA-512 [27] for the PRG, and an AEAD encryption scheme based on either SIV or a composition of CBC with HMAC [29, 30];
- **PRF-PRNG:** HKDF [21] with SHA-256 or SHA-512 [27], used as explained in Section 4.3.2.

We now detail the main differences:

Deferred randomness for sending. Deployed Signal implementations generate a new CKA message and absorb the resulting key into the RNG in Rcv, as opposed to taking care of this inside Send, as done here. The way it is done here is advantageous in the sense that the new key is not needed until the Send operation is actually initiated, so there is no need to risk its exposure unnecessarily (in case the state is compromised in between receiving and sending). Indeed, this security enhancement to Signal was explicitly mentioned by Marlinspike and Perrin [25] (cf. Section 6.5), and we simply follow this suggestion for better security.

Epoch indexing. In our scheme we have an explicit epoch counter t to index a given epoch. In Signal, one uses the uniqueness of latest CKA message (of the form g^x) to index an epoch. This saves an extra counter from each party's state, but we find our treatment of having explicit epoch counters much more intuitive, and not relying on any particular structure of CKA messages. In fact, indexing a dictionary becomes slightly more efficient when using a simple counter than the entire CKA message (which could be long for certain CKA protocols; e.g., post-quantum from lattices).

FS-AEAD abstraction. Unlike the SM proposed from this section, Signal does not use the FS-AEAD abstraction. Instead, each party maintains a sending and a receiving PRG that are kept in sync with the other party's receiving and sending PRG, respectively. Moreover, when receiving the first message of a new epoch, the current receive PRG is skipped ahead appropriately depending on the value ℓ, and the skipped keys are stored in a *single*, global dictionary. The state of the receive PRG is then overwritten with the new state output by the root RNG. Then, upon the next send operation new randomness for the CKA message is generated, and the sending RNG is also overwritten by the state output from updating the root RNG again. This is logically equivalent to our variant of Signal with the particular FS-AEAD implementation in Figure 6, except we will maintain multiple dictionaries (one for each epoch t). However, merging these dictionaries into one global dictionary (indexed by epoch counter in addition to the message count within epoch) becomes a simple efficiency optimization of the resulting scheme. Moreover, once this optimization is done, there is no need to store an array of FS-AEAD instances $v[t]$. Instead, we can only remember the latest sending and receiving FS-AEAD instance, overwriting them appropriately with each new epoch. Indeed, storing old message keys from not-yet-delivered messages is the only information

Signal Scheme

```
Init-A $(k)$
    id $\leftarrow$ A
    $(k_{\mathsf{root}}, k_{\mathsf{CKA}}) \leftarrow k$
    $\sigma_{\mathsf{root}} \leftarrow \mathsf{P\text{-}Init}(k_{\mathsf{root}})$
    $(\sigma_{\mathsf{root}}, w_{\mathsf{R}}) \leftarrow \mathsf{P\text{-}Up}(\sigma_{\mathsf{root}}, \lambda)$
    $\gamma \leftarrow \mathsf{CKA\text{-}Init\text{-}A}(k_{\mathsf{CKA}})$
    $T_{\mathsf{cur}} \leftarrow \lambda$
    $\ell_{\mathsf{prv}} \leftarrow 0$
    $t_{\mathsf{cur}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow 0$
    $\mathcal{D}[\cdot] \leftarrow \lambda$

skip $(t, u)$
    while $i_{\mathsf{R}} < u$
        $i_{\mathsf{R}}$ ++
        $(w_{\mathsf{R}}, K) \leftarrow G(w_{\mathsf{R}})$
        $\mathcal{D}[t, i_{\mathsf{R}}] \leftarrow K$

Send-A $(m)$
    if $t_{\mathsf{cur}}$ is even
        $t_{\mathsf{cur}}$ ++
        $\ell_{\mathsf{prv}} \leftarrow i_{\mathsf{S}}$
        $i_{\mathsf{S}} \leftarrow 0$
        $(\gamma, T_{\mathsf{cur}}, I) \leftarrow\$ \mathsf{CKA\text{-}S}(\gamma)$
        $(\sigma_{\mathsf{root}}, w_{\mathsf{S}}) \leftarrow \mathsf{P\text{-}Up}(\sigma_{\mathsf{root}}, I)$
    $i_{\mathsf{S}}$ ++
    $(w_{\mathsf{S}}, K) \leftarrow G(w_{\mathsf{S}})$
    $h \leftarrow (t_{\mathsf{cur}}, i_{\mathsf{S}}, T_{\mathsf{cur}}, \ell_{\mathsf{prv}})$
    $e \leftarrow \mathsf{Enc}(K, h, m)$
    return $(h, e)$

try-skipped $(t, i)$
    $K \leftarrow \mathcal{D}[t, i]$
    $\mathcal{D}[t, i] \leftarrow \perp$
    return $K$

Rcv-A $(c)$
    $(h, e) \leftarrow c$
    $(t, i, T, \ell) \leftarrow h$
    req $t$ even and $t \leq t_{\mathsf{cur}} + 1$
    if $t = t_{\mathsf{cur}} + 1$
        $\mathsf{skip}(t - 2, \ell)$
        $t_{\mathsf{cur}}$ ++, $i_{\mathsf{R}} \leftarrow 0$
        $(\gamma, I) \leftarrow \mathsf{CKA\text{-}R}(\gamma, T)$
        $(\sigma_{\mathsf{root}}, w_{\mathsf{R}}) \leftarrow \mathsf{P\text{-}Up}(\sigma_{\mathsf{root}}, I)$
    $K \leftarrow \mathsf{try\text{-}skipped}(t, i)$
    if $K = \perp$
        $\mathsf{skip}(t, i - 1)$
        $i_{\mathsf{R}}$ ++
        $(w_{\mathsf{R}}, K) \leftarrow G(w_{\mathsf{R}})$
    $m \leftarrow \mathsf{Dec}(K, h, e)$
    if $m = \perp$
        error
    return $(t, i, m)$
```

Figure 9: *Signal scheme without the FS-AEAD abstraction, based on a CKA scheme, a PRF-PRNG, authenticated encryption, and a regular PRG. The figure only shows the algorithms for* A*;* B*'s algorithms are analogous, with "even" replaced by "odd" (and with* Init-B *initializing* w_{S} *instead of* w_{R}*).*

one needs to remember from the prior FS-AEAD instances. So once this information is stored in the global dictionary, we can simply overwrite the remaining information when moving to the new epoch. With these simple efficiency optimizations, we arrive to (almost) precisely what is done by Signal (cf. Figure 9).

To sum up, blindly using the FS-AEAD abstraction results in a slightly less efficient scheme, but (1) we feel our treatment is more modular and intuitive; (2) when using a concrete FS-AEAD scheme from Section 4.2, getting actual Signal becomes a simple efficiency optimization of the resulting scheme. In particular, the security of Signal itself still follows from our framework.

Initial key agreement. As mentioned in the introduction, our modeling only addresses the double-ratchet aspect of the Signal protocol, and does not tackle the challenging problem of the generation of the initial shared key k. One thing this also allows us to do is to elegantly side-step the issue that natural CKA protocols are *unkeyed*, and do not generate shared a shared key I_0 from the initial message T_0. Instead, we model CKA as a *secret key* primitive, where the initial key k_{CKA} effectively generates the first message T_0 of "unkeyed CKA" protocol, but now shared keys $I_1, I_2, \ldots$ get generated right away from subsequent messages $T_1, T_2, \ldots$. In other words, rather than having k only store the root key k_{root}, in our protocol we let it store a tuple $(k_{\mathsf{root}}, k_{\mathsf{CKA}})$, and then use k_{CKA} to solve the syntactic issue of having a special treatment for the first CKA message T_0.

In most actual Signal implementations, the initial shared key k will only contain the value k_{root}, and it is the receiver B who stores several initial CKA messages T_0 (called "one-time prekeys") on the Signal server for new potential senders A. When such A comes along, A would take one such one-time prekey value T_0 from the Signal server, and (optionally) *use it* to generate the initial shared key k_{root} using the X3DH Key Agreement Protocol [26]. This creates slight circularity, and we leave it to the future work to properly model and analyze such generation of the initial key k_{root}.

5.3 Security of the SM Scheme

This section establishes SM security of the scheme presented in Section 5.1 by showing that it satisfies the correctness, authenticity, and privacy properties. By combining Theorems 7, 8, and 12 with Theorem 1, one obtains the following result:

Theorem 6. *Assume that*

- CKA *is a* $(t', \Delta_{\mathsf{CKA}}, \varepsilon_{\mathrm{cka}})$*-secure CKA scheme,*
- $\mathsf{FS\text{-}AEAD}$ *is a* $(t', q, \varepsilon_{\text{fs-aead}})$*-secure FS-AEAD scheme, and*
- P *is a* $(t', \varepsilon_{\mathrm{p}})$*-secure PRF-PRNG.*

Then, the SM *construction above is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$*-SM-secure for* $t \approx t'$*,* $\Delta_{\mathsf{SM}} = 2 + \Delta_{\mathsf{CKA}}$*, and*

$$\varepsilon \;\le\; 2q_{\mathsf{ep}}^2 \cdot (\varepsilon_{\mathrm{cka}} + q \cdot \varepsilon_{\text{fs-aead}} + \varepsilon_{\mathrm{p}}) \ .$$

The rest of the section is dedicated to proving Theorems 7, 8, and 12.

5.3.1 Correctness

Theorem 7. *Let* $t, q, q_{\mathsf{ep}} \in \mathbb{N}$*. The* SM *construction above is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, 0)$*-correct for* $\Delta_{\mathsf{SM}} = 2 + \Delta_{\mathsf{CKA}}$*, where* Δ_{CKA} *is the parameter of the underlying CKA scheme.*

Proof. Recall the correctness variant of the SM game allows no challenges and only provides reduced inject oracles, which only allow the attacker to deliver compromised messages pertaining to *non-current* epochs.

To argue that scheme SM is correct, note that it is straight-forward to verify that A and B produce the same CKA keys in this experiment and absorb them into their respective copies of the root RNG in the same order. Therefore, when A initializes a new (odd) epoch with a particular key, B will initialize the epoch on his end with the same key (and vice-versa). Moreover, it is not hard either to see that messages received are routed to the appropriate instance of FS-AEAD. Using the correctness of FS-AEAD, one concludes that scheme SM is correct as well. $\square$

5.3.2 Authenticity

SM scheme SM presented in Section 5.1 satisfies the authenticity property:

Theorem 8. *Assume that*

- CKA *is a* $(t', \Delta_{\mathsf{CKA}}, \varepsilon_{\mathrm{cka}})$*-secure CKA scheme,*
- $\mathsf{FS\text{-}AEAD}$ *is a* $(t', q, \varepsilon_{\text{fs-aead}})$*-secure FS-AEAD scheme, and*
- P *is a* $(t', \varepsilon_{\mathrm{p}})$*-secure PRF-PRNG.*

Then, the SM *construction above is* $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$*-authentic for* $t \approx t'$*,* $\Delta_{\mathsf{SM}} = 2 + \Delta_{\mathsf{CKA}}$*, and*

$$\varepsilon \;\le\; \varepsilon_{\text{cka}} + q \cdot \varepsilon_{\text{fs-aead}} + \varepsilon_{\text{p}} \ .$$

To prove the theorem, fix an attacker $\mathcal{A}$, and recall that $\mathcal{A}$ initially specifies values (t^*_{L}, t^*), where w.l.o.g. t^* is assumed to be *even* in the following. That is, in the epoch being attacked, B acts as sender and A as receiver. Recall further that the authenticity game provides no challenge oracles and inject oracles that are *reduced* (as defined in Section 3.3), except for ciphertexts corresponding to epoch t^*.[12] Moreover,

- if at any point $t_{\mathsf{L}} \in \{t^*_{\mathsf{L}} + 1, \ldots, t^* - 1\}$, the attacker loses the game immediately;
- if A (the receiver in epoch t^*) is corrupted any time once $t_{\mathsf{A}} > t^*_{\mathsf{L}}$, the attacker loses the game immediately;

In the following, let $t_{\mathsf{heal}} = t^*_{\mathsf{L}} + 2$.[13]

The proof that SM scheme SM presented in Section 5.1 satisfies the authenticity property considers hybrid experiments H_0, H_1, and H_2. The indistinguishability of H_0 and H_1 follows from the security of the CKA scheme, that of H_1 and H_2 from the security of the PRF-PRNG, and there is a reduction from winning the FS-AEAD security game to winning in H_2.

Hybrid H_0. This hybrid is the actual authenticity game for scheme SM.

Hybrid H_1. Denote by $T_1, T_2, \ldots$ and $I_1, I_2, \ldots$ the CKA messages and keys, respectively, created by A and B during the execution of the protocol. H_1 works as H_0, but CKA key $I_{t_{\mathsf{heal}}}$, is replaced by a value chosen uniformly at random.

Lemma 9. *There exists an attacker* $\mathcal{B}_1$ *(comparable in efficiency to* $\mathcal{A}$*) such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_0}(\mathcal{A}) \;\le\; \mathrm{Adv}^{\mathsf{SM}}_{H_1}(\mathcal{A}) + \mathrm{Adv}^{\mathsf{CKA}}_{\mathrm{ror},\Delta}(\mathcal{B}_1) \ .$$

Proof. Assume for simplicity that t_{heal} is even, i.e., the "healing" CKA message/key pair is produced by B. Attacker $\mathcal{B}_1$ (against the CKA game) internally runs $\mathcal{A}$ and simulates the authenticity game for $\mathcal{A}$. The simulation may fail under certain circumstances, but, as shown below, this only happens when $\mathcal{A}$ would lose the game anyway. $\mathcal{B}_1$ proceeds as follows:

- **init**(t^*_{L}, t^*): $\mathcal{B}_1$ calls its own oracle **init**(t_{heal}) and simulates initialization for both parties as in H_0, except that the CKA initialization procedures are not called (these are handled by the CKA game).
- **transmit-A**$(m, \perp)$: $\mathcal{B}_1$ simulates the oracle as in H_0, but, whenever needed, it creates CKA messages and keys by calling the corresponding oracle of the CKA game: to create (T_t, I_t) it calls **send-A**.
- **transmit-A**(m, r) for $r \neq \perp$: $\mathcal{B}_1$ simulates as in H_0, but it creates CKA messages and keys (T_t, I_t) by calling **send-A'**; if the call fails (due to the **req** condition of **send-A'**), $\mathcal{B}_1$ aborts.

[12] cf. Property (B) in Definition 7.

[13] Observe that this should indeed be $t_{\mathsf{heal}} = t^*_{\mathsf{L}} + 2$, and not $t^*_{\mathsf{L}} = t^*_{\mathsf{L}} + \Delta_{\mathsf{CKA}}$: following Section 4.1, all CKA schemes heal within two epochs; the parameter Δ_{CKA} merely measures the number of additional epochs needed for parties' states to become safe to corrupt.

- **transmit-B**$(m, \perp)$: $\mathcal{B}_1$ simulates as in H_0, but, whenever needed, it creates CKA messages and keys (T_t, I_t) by calling **chall-B** if $t = t_{\mathsf{heal}}$, and **send-B** otherwise.
- **transmit-B**(m, r) for $r \neq \perp$: $\mathcal{B}_1$ simulates as in H_0, but it creates CKA messages and keys (T_t, I_t) by calling **send-B'**; if the call fails (due to the **req** condition of **send-A'**), $\mathcal{B}_1$ aborts. If $t = t_{\mathsf{heal}}$, $\mathcal{B}_1$ also aborts.
- **deliver-A**(c) and **inject-A**(c): $\mathcal{B}_1$ simulates as in H_0 but calls **receive-A** (just to advance the CKA game) and uses the CKA keys I_t already returned by the CKA game (when $\mathcal{B}_1$ was handling a previous transmit query for B).
- **corr-A**: $\mathcal{B}_1$ simulates as in H_0, but uses the corruption oracle **corr-A** of the CKA game to get the state γ^{A}; if the call fails (due to the **req** condition of **corr-A**), $\mathcal{B}_1$ aborts.

The oracles corresponding to B not explicitly described above work analogously. In addition, as soon as a party reaches epoch $t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}}$, the reduction corrupts said party to obtain its state, which is used to complete the simulation.

The proof is completed by showing that whenever the bit inside the CKA game is $b = 0$ or $b = 1$, $\mathcal{B}_1$ simulates H_0 or H_1, respectively, to $\mathcal{A}$—up to the point where $\mathcal{A}$ wins or loses. This is verified by inspection and by noticing that that the restrictions of the authenticity game match the restrictions of the CKA game. In particular:

- $\mathcal{B}_1$ calls the CKA challenge oracle **chall-B** in epoch t_{heal}—as announced when $\mathcal{B}_1$ called **init**.
- Since $\mathcal{B}_1$ uses the leaked state to simulate once a party reaches $t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}}$, a call to **send-A'** (**send-B'**) only fails if the call to **transmit-A** (**transmit-B**) with $r \neq \perp$ causes an epoch-t message to be sent for $t \in \{t_{\mathsf{L}}^* + 1, \ldots, t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}}\}$. In such a case, due to

  $$t^* \geq t_{\mathsf{L}}^* + \Delta_{\mathsf{SM}} = t_{\mathsf{L}}^* + 2 + \Delta_{\mathsf{CKA}} = t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}} \ , \tag{1}$$

  the attacker loses the game because either

  - t_{L} is set to a value in $\{t_{\mathsf{L}}^* + 1, \ldots, t^* - 1\}$, or
  - t_{L} is set to t^* but before B sends any epoch-t^* messages, i.e., all epoch-t^* ciphertexts will be compromised and therefore the attacker cannot win the authenticity game.
- Due to (1), injections are only allowed once the CKA game has ended. Hence, the simulation only needs to handle forged CKA messages T once it is given the CKA state $(\gamma^{\mathsf{A}}, \gamma^{\mathsf{B}})$.
- Observe that if A is corrupted at any point after $t_{\mathsf{A}} > t_{\mathsf{L}}^*$, the attacker loses the game. If B is corrupted while $t_{\mathsf{B}} \in \{t_{\mathsf{L}}^* + 1, \ldots, t^* - 1\}$, $\mathcal{A}$ loses as well. If B gets corrupted in epoch t^* or later, the CKA game allows corrupting B as well, due to (1).

Hence, the simulation fails only if $\mathcal{A}$ would lose the authenticity game anyway. □

Hybrid H_2. Consider again CKA messages $T_1, T_2, \ldots$ and keys $I_1, I_2, \ldots$. Each time a party advances to a new epoch t, it absorbs I_t into its copy of the root RNG. Observe that once A, the receiver-to-be for epoch t^*, absorbs I_{t^*-1} and sends their first message in epoch $t^* - 1$, the attacker is able to start (non-trivial) injections for epoch t^* even though B, the sender-to-be for epoch t^*, may not have received any epoch-$(t^* - 1)$ message and thus still be in epoch $t^* - 2$. The only exception to this fact is for $\Delta_{\mathsf{CKA}} = 0$ and $t^* = t^*_{\mathsf{L}} + 2$: in this case the authenticity game does not allow the attacker to inject until both parties reach epoch t^*. This is important since the refreshing CKA key is I_{t^*} here.

Let σ_{t^*-1} be the state of the root RNG after the absorption of I_{t^*-1}. Observe that for each forged ciphertext c' with arbitrary CKA messages T', the attacker causes A to absorb some CKA key I' into the root RNG at state σ_{t^*-1}; the second component k' of the corresponding output $(\sigma', k') \leftarrow \mathsf{P\text{-}Up}(\sigma_{t^*-1}, I')$ is thereafter used to instantiate an FS-AEAD session that processes the FS-AEAD part of c'.

H_2 works as H_1, but each output (σ', k') of $\mathsf{P\text{-}Up}(\sigma_{t^*-1}, I')$ on some input I' is replaced by $(\sigma, \tilde{k}_{I'})$ for a randomly chosen value $\tilde{k}_{I'} \in \mathcal{R}$.

Lemma 10. *There exists an attacker $\mathcal{B}_2$ (comparable in efficiency to $\mathcal{A}$) such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_1}(\mathcal{A}) \;\leq\; \mathrm{Adv}^{\mathsf{SM}}_{H_2}(\mathcal{A}) + \mathrm{Adv}^{\mathsf{P}}_{\mathrm{PP}}(\mathcal{B}_2)\ .$$

Proof. Attacker $\mathcal{B}_2$ (against the PRF-PRNG game) internally runs $\mathcal{A}$ and simulates the authenticity game for $\mathcal{A}$. The simulation may fail under certain circumstances, but, as shown below, this only happens when $\mathcal{A}$ would lose the game anyway. $\mathcal{B}_2$ simulates as shown next. Observe that the simulation process potentially produces σ_{t^*}—the actual state of the RNG when B absorbs I_{t^*}. Once this happens, $\mathcal{B}_2$ uses this value to continue the simulation (while still using the PRF-PRNG game for injections pertaining to epoch t^* until A reaches epoch t^* as well).

- **init**(t^*_{L}, t^*): $\mathcal{B}_2$ simulates initialization for both parties as in H_2, except that the PRF-PRNG initialization procedures are not called (these are handled by the PRF-PRNG game).
- **transmit-A**(m, r): $\mathcal{B}_2$ simulates as in H_2, but it creates keys k for new FS-AEAD sessions as follows: If a key is needed during Send for epoch t, i.e., when $\mathsf{P\text{-}Up}$ is supposed to be called on state σ_{t-1} and some input I,
  - **process** is called *with no argument* for $t = t_{\mathsf{heal}}$ but $t \neq t^*$,
  - **chall-prng** is called *with no argument* for $t = t_{\mathsf{heal}}$ and $t = t^*$,
  - **chall-prf**(I) is called for $t \neq t_{\mathsf{heal}}$ but $t = t^*$, and
  - **process**(I) is called in all other cases.

  If **chall-prng** or **chall-prf** cannot be called (due to the **req** condition), $\mathcal{B}_2$ aborts.
- **deliver-A**(c): $\mathcal{B}_2$ simulates as in H_2 but uses the appropriate FS-AEAD key obtained from the PRF-PRNG game when processing the corresponding transmit query.
- **inject-A**(c): $\mathcal{B}_2$ simulates as in H_2, but when an FS-AEAD key for epoch t^* is needed, it creates it similarly to the transmit oracles above, by calling **chall-prf**(I).
- **corr-A**: $\mathcal{B}_2$ simulates as in H_2, but uses the corruption oracle of PRF-PRNG game to get the state σ. If this is not possible, due to $\neg\mathsf{prf}$ being false, $\mathcal{B}_2$ aborts.
- **corr-B**: $\mathcal{B}_2$ simulates as in H_2, but uses the corruption oracle of the PRF-PRNG game to get the state σ. If this is not possible, due to $\neg\mathsf{prf}$ being false, $\mathcal{B}_2$ proceeds as follows: if B is in epoch $t^* - 1$ or $t^* - 2$, it aborts; if B is in epoch t^* or later, use σ_{t^*} mentioned above or whatever state it has evolved to.

The oracles corresponding to B not explicitly described above work analogously. The proof is completed by showing that whenever bit inside the PRF-PRNG game is $b = 0$ or $b = 1$, $\mathcal{B}_2$ simulates H_1 or H_2, respectively, to $\mathcal{A}$—up to the point where $\mathcal{A}$ wins or loses. This is verified by inspection and by noticing that that the restrictions of the authenticity game match the restrictions of the PRF-PRNG game. In particular:

- Simulating the transmit oracles only fails if **chall-prng** or **chall-prf** cannot be called due to the **req** condition, which, implies that $\mathsf{corr} = \mathsf{true}$ when $t_{\mathsf{B}} = t^* - 1$. This, in turn, means that $t_{\mathsf{L}} \in \{t_{\mathsf{L}}^* + 1, \ldots, t^* - 1\}$, and thus the attacker would lose anyway.
- If simulating **corr-A** fails, $\mathsf{prf} = \mathsf{true}$, which only occurs once $t_{\mathsf{A}} = t^* - 1$, which implies that the corruption would set $t_{\mathsf{L}} \in \{t_{\mathsf{L}}^* + 1, \ldots, t^* - 1\}$, and thus the attacker would lose anyway.
- Simulating **corr-B** fails only if B is in epoch $t^* - 1$ or $t^* - 2$, which again implies that $\mathcal{A}$ would lose anyway.

□

Reduction to FS-AEAD. Observe that there are two ways for FS-AEAD session to be created for $t = t^*$:

- B creates his own when he advances to epoch t^*, or
- a session is created when $\mathcal{A}$ tries to inject towards A once she is in session $t^* - 1$.

In either case, in hybrid H_2, all keys generated for sessions with $t = t^*$ are uniformly random. Therefore, there is a straight-foroward hybrid argument, where in the i^{th} hybrid, attacking the first i FS-AEAD session with $t = t^*$ is not accepted as a win; each two successive hybrids are close by a straight-forward reduction to the FS-AEAD security; the proof is omitted.

Lemma 11. *There exists an attacker $\mathcal{B}_3$ (comparable in efficiency to $\mathcal{A}$) such that*

$$\mathrm{Adv}_{H_2}^{\mathsf{SM}}(\mathcal{A}) \;\leq\; q \cdot \mathrm{Adv}_{\text{fs-aead}}^{\mathsf{FS\text{-}AEAD}}(\mathcal{B}_3) \;.$$

5.3.3 Privacy

SM scheme SM presented in Section 5.1 satisfies the privacy property:

Theorem 12. *Assume that*

- CKA *is a $(t', \Delta_{\mathsf{CKA}}, \varepsilon_{\text{cka}})$-secure CKA scheme,*
- $\mathsf{FS\text{-}AEAD}$ *is a $(t', q, \varepsilon_{\text{fs-aead}})$-secure FS-AEAD scheme, and*
- P *is a $(t', \varepsilon_{\text{p}})$-secure PRF-PRNG.*

Then, the SM construction above is $(t, q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon)$-private for $t \approx t'$, $\Delta_{\mathsf{SM}} = 2 + \Delta_{\mathsf{CKA}}$, and

$$\varepsilon \;\leq\; \varepsilon_{\text{cka}} + q \cdot \varepsilon_{\text{fs-aead}} + \varepsilon_{\text{p}} \;.$$

To prove the theorem, fix an attacker $\mathcal{A}$, and recall that $\mathcal{A}$ initially specifies values (t_{L}^*, t^*), where w.l.o.g. t^* is assumed to be *even* in the following. That is, in the epoch being attacked, B acts as sender and A as receiver. Recall further that the privacy game provides only *reduced* inject oracles (as defined in Section 3.3) and that the challenge oracles only work in epoch t^*. Moreover, if at any point $t_{\mathsf{L}} \in \{t_{\mathsf{L}}^* + 1, \ldots, t^* - 1\}$, the attacker loses the game immediately (i.e., the outcome of the game is a uniformly random bit). In the following, let $t_{\mathsf{heal}} = t_{\mathsf{L}}^* + 2$.

Similarly, to authenticity, the proof that SM scheme SM presented in Section 5.1 satisfies the privacy property also considers hybrid experiments H_0, H_1, and H_2. Again, the indistinguishability of H_0 and H_1 follows from the security of the CKA scheme, that of H_1 and H_2 from the security of the PRF-PRNG, and there is a reduction from winning the FS-AEAD security game to winning in H_2.

Hybrid H_0. This hybrid is the actual privacy game for scheme SM.

Hybrid H_1. Denote by $T_1, T_2, \ldots$ and $I_1, I_2, \ldots$ the CKA messages and keys, respectively, created by A and B during the execution of the protocol. H_1 works as H_0, but CKA key $I_{t_{\mathsf{heal}}}$, is replaced by a value chosen uniformly at random.

Lemma 13. *There exists an attacker $\mathcal{B}_1$ (comparable in efficiency to $\mathcal{A}$) such that*

$$\mathrm{Adv}_{H_0}^{\mathsf{SM}}(\mathcal{A}) \;\leq\; \mathrm{Adv}_{H_1}^{\mathsf{SM}}(\mathcal{A}) + \mathrm{Adv}_{\mathrm{ror},\Delta}^{\mathsf{CKA}}(\mathcal{B}_1) \ .$$

Proof. Assume for simplicity that t_{heal} is even, i.e., the "healing" CKA message/key pair is produced by B. Attacker $\mathcal{B}_1$ (against the CKA game) internally runs $\mathcal{A}$ and simulates the privacy game for $\mathcal{A}$. The simulation may fail under certain circumstances, but, as shown below, this only happens when $\mathcal{A}$ would lose the game anyway. $\mathcal{B}_1$ proceeds as follows:

- **init**(t_{L}^*, t^*): $\mathcal{B}_1$ calls its own oracle **init**(t_{heal}) and simulates initialization for both parties as in H_0, except that the CKA initialization procedures are not called (these are handled by the CKA game).

- **transmit-A**$(m, \perp)$: $\mathcal{B}_1$ simulates the oracle as in H_0, but, whenever needed, it creates CKA messages and keys by calling the corresponding oracle of the CKA game: to create (T_t, I_t) it calls **send-A**.

- **transmit-A'**(m, r) for $r \neq \perp$: $\mathcal{B}_1$ simulates as in H_0, but it creates CKA messages and keys (T_t, I_t) by calling **send-A'**; if the call fails (due to the **req** condition of **send-A'**), $\mathcal{B}_1$ aborts.

- **chall-A**(m): $\mathcal{B}_1$ simulates as in H_0, i.e., the oracle offers no functionality since t^* is even.

- **transmit-B**$(m, \perp)$: $\mathcal{B}_1$ simulates as in H_0, but, whenever needed, it creates CKA messages and keys (T_t, I_t) by calling **chall-B** if $t = t_{\mathsf{heal}}$, and **send-B** otherwise.

- **transmit-B'**(m, r) for $r \neq \perp$: $\mathcal{B}_1$ simulates as in H_0, but it creates CKA messages and keys (T_t, I_t) by calling **send-B'**; if the call fails (due to the **req** condition of **send-A'**), $\mathcal{B}_1$ aborts. If $t = t_{\mathsf{heal}}$, $\mathcal{B}_1$ also aborts.

- **deliver-A**(c) and **inject-A**(c): $\mathcal{B}_1$ simulates as in H_0 but calls **receive-A** (just to advance the CKA game) and uses the CKA keys I_t already returned by the CKA game (when $\mathcal{B}_1$ was handling a previous transmit query for B).

- **corr-A**: $\mathcal{B}_1$ simulates as in H_0, but uses the corruption oracle **corr-A** of the CKA game to get the state γ^{A}; if the call fails (due to the **req** condition of **corr-A**), $\mathcal{B}_1$ aborts.

The oracles corresponding to B not explicitly described above work analogously. In addition, as soon as a party reaches epoch $t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}}$, the reduction corrupts said party to obtain its state, which is used to complete the simulation.

The proof is completed by showing that whenever the bit inside the CKA game is $b = 0$ or $b = 1$, $\mathcal{B}_1$ simulates H_0 or H_1, respectively, to $\mathcal{A}$—up to the point where $\mathcal{A}$ wins or loses. This is verified by inspection and by noticing that that the restrictions of the authenticity game match the restrictions of the CKA game. In particular:

- $\mathcal{B}_1$ calls the CKA challenge oracle **chall-B** in epoch t_{heal}—as announced when $\mathcal{B}_1$ called **init**.
- Since $\mathcal{B}_1$ uses the leaked state to simulate once a party reaches $t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}}$, a call to **send-A'** (**send-B'**) only fails if the call to **transmit-A** (**transmit-B**) with $r \neq \perp$ causes an epoch-t message to be sent for $t \in \{t^*_{\mathsf{L}} + 1, \ldots, t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}}\}$. In such a case, due to

$$t^* \geq t^*_{\mathsf{L}} + \Delta_{\mathsf{SM}} = t^*_{\mathsf{L}} + 2 + \Delta_{\mathsf{CKA}} = t_{\mathsf{heal}} + \Delta_{\mathsf{CKA}} \ , \tag{2}$$

  the attacker either loses the game because t_{L} is set to a value in $\{t^*_{\mathsf{L}} + 1, \ldots, t^* - 1\}$, or t_{L} is set to t^* but before B sends any epoch-t^* messages, i.e., all epoch-t^* ciphertexts will be compromised and therefore the attacker cannot ask for any challenges (and will have no information about the secret bit b).
- Observe that if either party is corrupted while $t_{\mathsf{B}} \in \{t^*_{\mathsf{L}} + 1, \ldots, t^* - 1\}$, $\mathcal{A}$ loses. If B gets corrupted in epoch t^* or later, the CKA game allows corrupting B as well, due to (2). However, A may get corrupted when she is in epoch $t^* - 1$ (before having received any epoch-t^* message) but B is already in t^*. At that point the CKA game does not allow corruption, but the fact that the privacy game allowed the corruption in the first place implies that no challenge has been sent yet. Hence, all remaining epoch-t^* ciphertexts will be compromised and therefore the attacker cannot ask for any challenges (and will have no information about the secret bit b). Once A reaches t^*, the CKA game allows corruption as well, again due to (2).

Hence, the simulation fails only if $\mathcal{A}$ would lose the authenticity game anyway. □

Hybrid H_2. Consider again CKA messages $T_1, T_2, \ldots$ and keys $I_1, I_2, \ldots$. Each time a party advances to a new epoch t, it absorbs I_t into its copy of the root RNG. For arbitrary i, let σ_i be the state of the root RNG after the absorption of I_i. H_2 works as H_1, but the output (σ_{t^*}, k_{t^*}) of $\mathsf{P\text{-}Up}(\sigma_{t^*-1}, I_{t^*})$ is replaced by $(\sigma_{t^*}, \tilde{k}_{t^*})$ for a randomly chosen value $\tilde{k}_{t^*} \in \mathcal{R}$.

Lemma 14. *There exists an attacker $\mathcal{B}_2$ (comparable in efficiency to $\mathcal{A}$) such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_1}(\mathcal{A}) \ \leq \ \mathrm{Adv}^{\mathsf{SM}}_{H_2}(\mathcal{A}) + \mathrm{Adv}^{\mathsf{P}}_{\mathrm{PP}}(\mathcal{B}_2) \ .$$

Proof. Attacker $\mathcal{B}_2$ (against the PRF-PRNG game) internally runs $\mathcal{A}$ and simulates the authenticity game for $\mathcal{A}$. The simulation may fail under certain circumstances, but, as shown below, this only happens when $\mathcal{A}$ would lose the game anyway. $\mathcal{B}_2$ proceeds as follows:

- **init**(t^*_{L}, t^*): $\mathcal{B}_2$ simulates initialization for both parties as in H_2, except that the PRF-PRNG initialization procedures are not called (these are handled by the PRF-PRNG game).

- **transmit-A**(m, r): $\mathcal{B}_2$ simulates as in H_2, but it creates keys k for new FS-AEAD sessions as follows: If a key is needed during Send for epoch t, i.e., when $\mathsf{P\text{-}Up}$ is supposed to be called on state σ_{t-1} and input I_t,
  - **process** is called *with no argument* for $t = t_{\mathsf{heal}}$ but $t \neq t^*$,
  - **chall-prng** is called *with no argument* for $t = t_{\mathsf{heal}}$ and $t = t^*$,
  - **chall-prng**(I) is called for $t \neq t_{\mathsf{heal}}$ but $t = t^*$, and
  - **process**(I) is called in all other cases.

  If **chall-prng** cannot be called (due to the **req** condition), $\mathcal{B}_2$ aborts.
- **deliver-A**(c): $\mathcal{B}_2$ simulates as in H_2 but uses the appropriate FS-AEAD key obtained from the PRF-PRNG game when processing the corresponding transmit query.
- **inject-A**(c): $\mathcal{B}_2$ simulates as in H_2.
- **corr-A**: $\mathcal{B}_2$ simulates as in H_2, but uses the corruption oracle of PRF-PRNG game to get the state σ.

The oracles corresponding to B not explicitly described above work analogously. The proof is completed by showing that whenever bit inside the PRF-PRNG game is $b = 0$ or $b = 1$, $\mathcal{B}_2$ simulates H_1 or H_2, respectively, to $\mathcal{A}$—up to the point where $\mathcal{A}$ loses. This is verified by inspection and by noticing that that the restrictions of the authenticity game match the restrictions of the PRF-PRNG game. In particular, simulating the transmit oracles only fails if **chall-prng** cannot be called due to the **req** condition, which, implies that $\mathsf{corr} = \mathsf{true}$ when $t_{\mathsf{B}} = t^* - 1$. This, in turn, means that $t_{\mathsf{L}} \in \{t^*_{\mathsf{L}} + 1, \ldots, t^* - 1\}$, and thus the attacker would lose anyway.

□

Reduction to FS-AEAD. Observe that in hybrid H_2, the key generated for FS-AEAD session $t = t^*$ is uniformly random. Therefore, there is a straight-foroward hybrid argument, where in the i^{th} hybrid, attacking the first i FS-AEAD session with $t = t^*$ is not accepted as a win; each two successive hybrids are close by a straight-forward reduction to the FS-AEAD security; the proof is omitted.

Lemma 15. *There exists an attacker $\mathcal{B}_3$ (comparable in efficiency to $\mathcal{A}$) such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_2}(\mathcal{A}) \;\leq\; q \cdot \mathrm{Adv}^{\mathsf{FS\text{-}AEAD}}_{\text{fs-aead}}(\mathcal{B}_3) \ .$$

6 On Stronger Security Notions

In the past two years several stronger notions of security beyond what is achieved by our secure messaging protocol. A selection of them and their relation to this work is presented below.

6.1 Security against Fine-Grained Compromise

A concern with the Signal protocol one might raise is that the states of both parties are identical. In other words, corrupting party, say, A reveals the secrets of party B (at least once all messages in transit from A to B are delivered) since the secrets are symmetric. In particular, if A and B are executing the Signal protocol, then corrupting A allows to break the following security properties (amongst others) during the current epoch:

(1) Privacy of future messages sent by A.

(2) Authenticity of future messages sent by A.

(3) Privacy of future message sent by B.

(4) Authenticity of future messages sent by B.

Given that, by correctness, A should be able use her (leaked) state to decrypt future messages from B, it seems that losing property (3) is unavoidable. Moreover, once A's state leaks, she holds no secret that is not already known to the adversary (until the next epoch begins), and thus giving up on (2) is also unavoidable. However, no such arguments exist for properties (1) and (4). Indeed, these and various more involved security properties are considered under the general moniker of security against *fine-grained state compromise* in [18]. Similarly, in [28] the authors also define a stronger notion of security for such channels which captures security against these and other more advanced attacks. Furthermore, in both papers new protocols satisfying the respective new security notions are presented.

A high price. A major appeal of the Signal protocol is its real-world practicality: besides allowing for immediate decryption, it also uses only a small amount of local storage to keep state, is built from a collection of standard and widely implemented primitives, and requires quite a limited amount of computation for any given operation. Unfortunately, to achieve their new security notions, the constructions of [18, 28] drop immediate decryption and make use of much heavier cryptographic machinery than the original Signal protocol. For example, both the key-updatable KEMs of [28] and the forward-secure public-key encryption of [18] are built from *hierarchical identity based encryption (HIBE)* and their states grow linearly in the number of messages exchanged during an epoch. This results in significantly larger states, more computation, added communication complexity (e.g., due to including public keys for a forward-secure signature scheme as well as forward-secure encryption scheme with each ciphertext sent [18]), and makes use of the relatively non-standard HIBE primitive (itself based on computationally expensive pairings or lattices).

While, for some use cases, this may be an acceptable price to pay for the added security, we believe that there remain a large number of realistic settings where the added cost is too much to ask. For example:

- when building a secure message platform intended to support very cheap (and thus under-resourced) smartphones;
- when the system is expected to still allow for some communication despite potential packet loss and/or re-ordered messages (e.g., due to an unreliable network);

- when lacking the (considerable) resources and cryptographic engineering know-how required to securely implement, test, and audit new advanced cryptographic primitives on all natively supported devices.

To address these and similar cases, we describe a compromise protocol in Section 6.2. While the new protocol strengthens the security of Signal (e.g., it protects properties (1) and (4) above), it does not give up immediate decryption, only makes use of standard cryptographic primitives constructions, which are already widely implemented in standard cryptographic libraries, and requires only a modest increase in local storage, computation, and bandwidth compared to the original protocol. Formally capturing the security notion achieved by the protocol as well as a corresponding security proof are left for future work.

6.2 Public-Key Secure Messaging

In a nutshell, the key idea is to extend the SM protocol from Section 5 and to break the symmetry between the internal states of A and B by using CCA2-secure public-key encryption (PKE) and EUF-CMA-secure digital signatures (DS)—standard public-key primitives—to encrypt and sign, respectively, the ciphertexts produced by the FS-AEAD scheme. Furthermore, with every message sent, a party "announces" encryption and verification keys for future epochs. Specifically, when sending epoch-t messages,[14] A attaches the encryption key with which B would encrypt in epoch $t+1$ as well as the verification key corresponding to the signing key A will use in epoch $t+2$.

In the following, let $\mathsf{PKE} = (\mathsf{PKEG}, \mathsf{Enc}, \mathsf{Dec})$ be a CCA2-secure PKE scheme and $\mathsf{DSS} = (\mathsf{DSG}, \mathsf{Sign}, \mathsf{Verify})$ a EUF-CMA-secure DSS (cf. Section 2).

State. Scheme SM keeps an internal state s_A (resp. s_B), which is initialized by $\mathsf{Init\text{-}A}$ (resp. $\mathsf{Init\text{-}B}$) and used as well as updated by Send and Rcv. Compared to the SM scheme from Section 5, the state s_A of SM consists of the following additional values (the state s_B is defined analogously):

- dictionaries $\mathsf{ek}[\cdot]$ and $\mathsf{dk}[\cdot]$ storing encryption and decryption keys, respectively, for public-key encryption, and
- dictionaries $\mathsf{sk}[\cdot]$ and $\mathsf{vk}[\cdot]$ storing signing and verification keys, respectively, for digital signatures.

By convention, epoch-t messages are encrypted with $\mathsf{ek}[t]$ and signed with $\mathsf{sk}[t]$. At any given time, the two parties store only a subset of all these keys:

- encryption key $\mathsf{ek}[t]$ and signing key $\mathsf{sk}[t]$ for the current sending epoch t;
- decryption keys $\mathsf{dk}[t]$ and verification keys $\mathsf{vk}[t]$ for all receiving epochs t for which not all messages have been received (i.e., for which $\mathsf{FS\text{-}Cnt}(v[t]) < \mathcal{M}[t]$).

[14] Recall that A sends messages in odd epochs and receives messages in even epochs.

Public Key Secure-Messaging (PKSM) Scheme

```
Init-A $(k')$
    id $\leftarrow$ A
    $(k, \mathsf{PKEK}, \mathsf{DSK}) \leftarrow k'$
    $(k_{\mathsf{root}}, k_{\mathsf{CKA}}) \leftarrow k$
    $\mathsf{ek}_0, \mathsf{dk}_0, \mathsf{ek}_1, \mathsf{dk}_1 \leftarrow \mathsf{PKEK}$
    $\mathsf{sk}_0, \mathsf{vk}_0, \dots, \mathsf{sk}_2, \mathsf{vk}_2 \leftarrow \mathsf{DSK}$

    $\sigma_{\mathsf{root}} \leftarrow \mathsf{P\text{-}Init}(k_{\mathsf{root}})$
    $(\sigma_{\mathsf{root}}, k) \leftarrow \mathsf{P\text{-}Up}(\sigma_{\mathsf{root}}, \lambda)$
    $v[\cdot] \leftarrow \lambda$
    $v[0] \leftarrow \mathsf{FS\text{-}Init\text{-}R}(k)$
    $\gamma \leftarrow \mathsf{CKA\text{-}Init\text{-}A}(k_{\mathsf{CKA}})$
    $T_{\mathsf{cur}} \leftarrow \lambda$
    $\mathsf{ek}[\cdot], \mathsf{dk}[\cdot], \mathsf{sk}[\cdot], \mathsf{vk}[\cdot] \leftarrow \lambda$
    $\mathsf{dk}[0] \leftarrow \mathsf{dk}_0$, $\mathsf{vk}[0] \leftarrow \mathsf{vk}_0$
    $\mathsf{ek}[1] \leftarrow \mathsf{ek}_1$, $\mathsf{sk}[1] \leftarrow \mathsf{sk}_1$
    $\mathsf{dk}[2] \leftarrow \mathsf{dk}_2$, $\mathsf{vk}[2] \leftarrow \mathsf{vk}_2$
    $\ell_{\mathsf{prv}} \leftarrow 0$
    $t_{\mathsf{cur}} \leftarrow 0$

Send-A $(m)$
    **if** $t_{\mathsf{cur}}$ *is even*
        $\ell_{\mathsf{prv}} \leftarrow \mathsf{FS\text{-}Stop}(v[t_{\mathsf{cur}} - 1])$
        $t_{\mathsf{cur}}$ ++
        $(\gamma, T_{\mathsf{cur}}, I) \leftarrow\$ \mathsf{CKA\text{-}S}(\gamma)$
        $(\sigma_{\mathsf{root}}, k) \leftarrow \mathsf{P\text{-}Up}(\sigma_{\mathsf{root}}, I)$
        $v[t_{\mathsf{cur}}] \leftarrow \mathsf{FS\text{-}Init\text{-}S}(k)$
        $(\mathsf{ek}[t_{\mathsf{cur}} + 1], \mathsf{dk}[t_{\mathsf{cur}} + 1]) \leftarrow\$ \mathsf{PKEG}$

        $(\mathsf{sk}[t_{\mathsf{cur}} + 2], \mathsf{vk}[t_{\mathsf{cur}} + 2]) \leftarrow\$ \mathsf{DSG}$

    $h \leftarrow$
      $(t_{\mathsf{cur}}, \ell_{\mathsf{prv}}, T_{\mathsf{cur}}, \mathsf{ek}[t_{\mathsf{cur}} + 1], \mathsf{vk}[t_{\mathsf{cur}} + 2])$

    $(v[t_{\mathsf{cur}}], e') \leftarrow \mathsf{FS\text{-}Send}(v[t_{\mathsf{cur}}], h, m)$
    $e \leftarrow \mathsf{Enc}(\mathsf{ek}[t_{\mathsf{cur}}], e')$
    $s \leftarrow \mathsf{Sign}(\mathsf{sk}[t_{\mathsf{cur}}], (h, e))$
    **return** $(h, e, s)$

Rcv-A $(c)$
    $(h, e, s) \leftarrow c$
    $(t, \ell, T, \mathsf{ek}, \mathsf{vk}) \leftarrow h$
    **req** $t$ even and $t \leq t_{\mathsf{cur}} + 1$
    **if** $\neg\mathsf{Verify}(\mathsf{vk}[t], (h, e), s)$
        **error**
    $e' \leftarrow \mathsf{Dec}(\mathsf{dk}[t], e)$
    **if** $t = t_{\mathsf{cur}} + 1$
        $t_{\mathsf{cur}}$ ++
        $\mathsf{FS\text{-}Max}(v[t - 2], \ell)$
        $\mathsf{sk}[t - 1], \mathsf{ek}[t], \mathsf{vk}[t + 1] \leftarrow \lambda$
        $\mathsf{ek}[t + 1] \leftarrow \mathsf{ek}$, $\mathsf{vk}[t + 2] \leftarrow \mathsf{vk}$
        $(\gamma, I) \leftarrow \mathsf{CKA\text{-}R}(\gamma, T)$
        $(\sigma_{\mathsf{root}}, k) \leftarrow \mathsf{P\text{-}Up}(\sigma_{\mathsf{root}}, I)$
        $v[t] \leftarrow \mathsf{FS\text{-}Init\text{-}R}(k)$
    $(v[t], i, m) \leftarrow \mathsf{FS\text{-}Rcv}(v[t], h, e')$
    **if** $m = \perp$
        **error**
    **return** $(t, i, m)$
```

Figure 10: *Secure-messaging scheme based on an FS-AEAD, a CKA scheme, a CCA secure public key encryption scheme, an EUF-CMA secure digital signature scheme and a PRF-PRNG. The figure only shows the algorithms for* A*;* B*'s algorithms are analogous, with "even" replaced by "odd" (and with* Init-B *initializing* $v[0]$ *via* FS-Init-S *as well as storing the "opposite" public/secret keys).*

The algorithms. The algorithms of the PKSM scheme are depicted in Figure 10 and described in more detail below; the differences to the original SM scheme in Figure 8 are highlighted. For ease of description, the algorithms Send and Rcv are presented as Send-A and Rcv-A, which handle the case where id = A; the case id = B works analogously. Moreover, to improve readability, the state s_{A} is not made explicit in the description: it consists of the variables set by the initialization algorithm (except for the four lines describing how to parse the initial shared key).

- **Initialization:** The initialization procedure Init-A expects a key k' shared between A and B; k is assumed to have been created at some point before the execution during a trusted setup phase and to consist of
  - initialization keys $k = (k_{\mathsf{root}}, k_{\mathsf{CKA}})$ for the root RNG and the CKA scheme, respectively;
  - PKE key pairs PKEK for epochs 0 and 1; and
  - DS key paris DSK for epochs 0, 1, and 2.

  The parties then store the appropriate keys (as described above) in their dictionaries $\mathsf{ek}[\cdot]$, $\mathsf{dk}[\cdot]$, $\mathsf{sk}[\cdot]$, and $\mathsf{vk}[\cdot]$. The remainder of the initialization is identical to that of the original SM scheme.

The PKSM protocol uses the PRF-PRNG, CKA, and FS-AEAD similarly to the original SM protocol, except for the modifications described above.

- **Sending messages:** Procedure $\mathsf{Send\text{-}A}$ allows A to send a message to B. If A enters a new sending epoch t, the protocol proceeds as before, but additionally generates the PKE key pair $(\mathsf{ek}[t+1], \mathsf{dk}[t+1])$ for epoch $t+1$ and the DS key pair $(\mathsf{sk}[t+2], \mathsf{vk}[t+2])$ for epoch $t+2$.

  Irrespective of whether A entered a new sending epoch, in addition to the values t_{A}, ℓ_{prv}, and T_{cur}, the header h now also includes the keys $\mathsf{ek}[t_{\mathsf{A}}+1]$ and $\mathsf{vk}[t_{\mathsf{A}}+2]$. Furthermore, the FS-AEAD ciphertext e' is additionally encrypted with $\mathsf{ek}[t_{\mathsf{A}}]$ and signed with $\mathsf{sk}[t_{\mathsf{A}}]$; observe that $\mathsf{ek}[t_{\mathsf{A}}]$ is either part of the initial shared key or has been previously announced by B, and, similarly, $\mathsf{sk}[t_{\mathsf{A}}]$ is either part of the initial shared key or has been previously generated by A. The resulting ciphertext e and signature s as well as h make up the eventual ciphertext.

- **Receiving messages:** When $\mathsf{Rcv\text{-}A}$ processes a ciphertext $c = (h, e, \ell)$ with header $h = (t, \ell, T, \mathsf{ek}, \mathsf{vk})$, it first verifies the signature with $\mathsf{vk}[t]$, which is either part of the initial shared key or has been previously announced by B. If the check fails, the algorithm throws an exception (**error**) and terminates; otherwise, it uses decryption key $\mathsf{dk}[t]$, which is either part of the initial shared key or has been generated earlier by A, to decrypt e, yielding an FS-AEAD ciphertext e'.

  As with the original SM scheme, there are two possibilities:

  - $t \leq t_{\mathsf{A}}$ (and t even): In this case, ciphertext c pertains to an existing FS-AEAD epoch, in which case the $\mathsf{Rcv\text{-}A}$ proceeds as before, but additionally deletes $\mathsf{dk}[t]$ and $\mathsf{vk}[t]$ if all messages for epoch t have been received.
  - $t = t_{\mathsf{A}} + 1$ and t_{A} odd: Again, the algorithm proceeds as before, except that it
    - deletes $\mathsf{sk}[t-1]$, $\mathsf{ek}[t]$, and $\mathsf{vk}[t+1]$ (as there is no longer any need to sign epoch-$(t-1)$ messages nor to announce the encryption key for epoch t or the verification key for epoch $t+2$) and
    - stores ek as the encryption key for epoch $t+1$ and vk as the verification key for epoch $t+2$.

Efficiency. In comparison with the original protocol in Figure 8 the packets being exchanged are significantly larger as they now each contain two extra public keys, a signature, and a public key encryption of the AEAD ciphertext. In practice, this may prove to be a high price to pay in bandwidth costs for the added security.

Fortunately, there are various optimizations one could apply to our scheme to reduce some of these overheads. We leave the study of these optimizations to future work, here only mentioning some high-level ideas. First, we might be able to reduce the size of two public-keys to a single public key by using a "signature-encryption" scheme (e.g., the ones in [16]), which allows to securely reuse the same key material for "compatible" signature and encryption schemes. In addition to bandwidth of each packet (now only one public-key), this even saves on the secret state of a party, as the same secret key will be used both for signing in one epoch, and decrypting in the next one.

While using a signature-encryption scheme does lead to smaller packet size due to only one public key, such schemes do not make any effort to compress ciphertexts and signatures themselves, nor to reduce the time for sender's signing/encrypting (resp. receiver's verifying/decrypting). For this task, a better suited primitive might be a signcryption scheme [32]. Indeed, the main original motivation for signcryption was to be more efficient (in time, "signcryptext" size, and/or public key size) as compared to the sum of separate signature and encryption schemes.

In fact, even more optimizations might be possible in a specific context, since for our application, each party will use a given signcryption key only in two epochs: as a sender with one particular recipient in one epoch, and as receiver with a one particular sender in another. It might be possible that such restricted signcryption schemes could be even more efficient than general ones.

Finally, a naive computation of AEAD, signature and public-key encryption will require multiple passes over a potentially long message. Thus, further optimizations should be possible, so that only a single pass over the message can compute all three functionalities.

Security intuition. Intuitively, the new scheme inherits the SM security properties of the original protocol in Figure 8 since the adversary's job has, if anything, only become harder. That is, even if both the public key encryption and signature schemes were trivially breakable it still remains for the adversary to attack the original protocol. However, now, when corrupting A the adversary does not learn the current signing key of B, nor does it learn B's current decryption key. Thus (assuming the encryption and signature schemes are secure), A's state cannot be used to decrypt future messages sent by A in the current epoch t (as they are encrypted with $\mathsf{ek}[t]$) nor can the adversary forge messages from B to A since these would have to verify under the corresponding verification key. We leave a formal analysis of the security achieved here (as well as the variant using signcryption directly) for future work.

Limits of the construction. It is worth noting that although the modified protocol does satisfy a stronger security notion than SM security in Figure 2 it does not achieve the very powerful notions of [18, 28]. For example, the notion in [18] also requires security in the following sense. Consider a single epoch in which A has sent a messages but which has not yet been delivered to B. Now the adversary corrupts both A and B. Then [18] require that the adversary should not be able to forge a different first message for that epoch than the one already sent out by A.

Clearly, even our modified scheme above does not satisfy this notion since the adversary learns the symmetric key needed to create an appropriate FS-AEAD from B while she learns the required signing (and public key encryption) keys from A. In contrast, the protocol of [18] essentially uses forward-secure signatures instead of our plain signatures. Thus, once A signs a first message for the epoch she no longer posses the signing key material to sign any other first message (or ciphertext really) for that epoch. While it is relatively immediate that using forward-secure signatures would also be an option for our protocol so as to get security against the above (and similar) attack we believe that the added practical costs could often outweigh the benefits. (A similar, but even more stark trade-off can be observed by replacing our plain public key encryption with forward secure variants). With an eye towards practicality, for the reasons described above, we have opted for the less secure but simpler and more lightweight variant above.

6.3 RECOVER Security

In recent concurrent and independent works by Durak and Vaudenay [11] and Jost, Maurer and Mularczyk [19], the authors introduce a very interesting notion of RECOVER-security, which, intuitively, ensures that once an adversary manages to forge a message from A to B, then no future message from A to B will be accepted by B. The converse must also hold when switching the roles of A and B. In a nutshell, to achieve this notion, their secure messaging protocol includes $h = H(trans)$ as part of the associated data (AD) of each new ciphertext being sent. Here H is a collision resistant hash function and $trans$ is a list containing all AD/ciphertexts pairs sent thus far. To see why this achieves RECOVER-security notice that once B accepts a forged message, i.e. one not sent by A, their values of *trans* will hence forth always differ. Thus any future message A sends to B will not have the expected value of h and so decryption will fail on B's end.

It is tempting to apply the same idea to Signal (and our extension in Figure 8), but notice that the naive application of hashing the entire transcript comes in tension with immediate decryption (which the protocol of [11, 19] does not meet), as the receiver may receive messages out of order, and not have the full transcript to hash. Moreover, if one cares about message loss recovery (when messages can be lost forever), then this idea will not work. We leave the problem adding meaningful and achievable form of RECOVER-security (especially in relation to immediate decryption) to our notions and protocols as an interesting problem for future research.

References

[1] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium.*, pages 327–343. USENIX Association, 2016.

[2] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 619–650. Springer, 2017.

[3] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, Proceedings*, pages 1–18, 2003.

[4] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, October 28, 2004*, pages 77–84, 2004.

[5] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, Report 2017/634, 2017. `https://eprint.iacr.org/2017/634`.

[6] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018, 2016.

[7] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017*, pages 451–466. IEEE, 2017.

[8] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016*, pages 164–178. IEEE Computer Society, 2016.

[9] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-rtt key exchange. In *Advances in Cryptology - EUROCRYPT 2018, Proceedings, Part III*, pages 425–455, 2018.

[10] Nir Drucker and Shay Gueron. Continuous key agreement with reduced bandwidth. Cryptology ePrint Archive, Report 2019/088, 2019. `https://eprint.iacr.org/2019/088`.

[11] F. Betül Durak and Serge Vaudenay. Bidirectional asynchronous ratcheted key agreement without key-update primitives. Cryptology ePrint Archive, Report 2018/889, 2018. `https://eprint.iacr.org/2018/889`.

[12] Messenger secret conversations: Technical whitepaper. `https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf`.

[13] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *Advances in Cryptology - ASIACRYPT 2002, Proceedings*, pages 548–566, 2002.

[14] Matthew D. Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 305–320, 2015.

[15] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-rtt key exchange with full forward secrecy. In *Advances in Cryptology - EUROCRYPT 2017, Proceedings, Part III*, pages 519–548, 2017.

[16] Stuart Haber and Benny Pinkas. Securely combining public-key cryptosystems. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 215–224. ACM, 2001.

[17] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *CHES 2017*, pages 232–252, 2017.

[18] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In *CRYPTO 2018, Part I*, pages 33–62, 2018.

[19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. EUROCRYPT 2019, to appear. `https://eprint.iacr.org/2018/954`.

[20] Dr. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997.

[21] Dr. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.

[22] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic Curves for Security. RFC 7748, January 2016.

[23] Joshua Lund. Signal partners with microsoft to bring end-to-end encryption to skype. `https://signal.org/blog/skype-partnership/`.

[24] M. Marlinspike. Open whisper systems partners with google on end-to-end encryption for allo. `https://signal.org/blog/allo/`.

[25] M. Marlinspike and T. Perrin. The double ratchet algorithm, 11 2016. `https://whispersystems.org/docs/specifications/doubleratchet/doubleratchet.pdf`.

[26] M. Marlinspike and T. Perrin. The double ratchet algorithm, 11 2016. `https://signal.org/docs/specifications/x3dh/x3dh.pdf`.

[27] National Institute of Standards and Technology (NIST). FIPS 180-4. secure hash standard. Technical report, US Department of Commerce, Aug 2015.

[28] Bertram Poettering and Paul Rösler. Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296, 2018. `https://eprint.iacr.org/2018/296`.

[29] Phillip Rogaway. Authenticated-encryption with associated-data. In *CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107, 2002.

[30] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT 2006, May 28 - June 1, 2006, Proceedings*, pages 373–390, 2006.

[31] Whatsapp encryption overview: Technical white paper, December 2017. `https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf`.

[32] Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) $<<$ cost(signature) + cost(encryption). In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 165–179, 1997.

A Simplified Properties Imply Full SM Security

Theorem 1. *Assume an SM scheme* SM *is*

- $(t', q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon_{\text{corr}})$*-correct,*
- $(t', q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon_{\text{auth}})$*-authentic, and*
- $(t', q, q_{\mathsf{ep}}, \Delta_{\mathsf{SM}}, \varepsilon_{\text{priv}})$*-private.*

Then, it is also $(t, \Delta_{\mathsf{SM}}, q, q_{\mathsf{ep}}, \varepsilon)$*-secure, where*

$$\varepsilon \;\leq\; \varepsilon_{\text{corr}} + q_{\mathsf{ep}}^2 \cdot (\varepsilon_{\text{auth}} + \varepsilon_{\text{priv}}) \;.$$

In order to prove Theorem 1, fix Δ_{SM} and a (t, q, q_{ep})-attacker $\mathcal{A}$. The proof considers a series of hybrids to arrive at an experiment in which the attacker has advantage 0. The distances between successive hybrids are bounded via reductions to correctness, authenticity, and privacy.

Hybrid H_0. Hybrid H_0 is the original SM game. For notational purposes, denote the advantage of $\mathcal{A}$ by $\mathrm{Adv}^{\mathsf{SM}}_{H_0}(\mathcal{A})$.

A.1 Dealing with Correctness

Hybrid H_1. Hybrid H_1 works as H_0, except that the **win**-condition inside the delivery oracles **deliver-A** and **deliver-B** is removed.

Lemma 16. *There exists a* (t', q, q_{ep})*-attacker* $\mathcal{B}_1$ *with* $t' \approx t$ *such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_0}(\mathcal{A}) \;\leq\; \mathrm{Adv}^{\mathsf{SM}}_{H_1}(\mathcal{A}) + \mathrm{Adv}^{\mathsf{SM}}_{\text{corr},\Delta_{\mathsf{SM}}}(\mathcal{B}_1) \;.$$

Proof. For both experiments H_0 and H_1, consider the event $\mathcal{E}$ that the **win**-condition inside one of the delivery oracles is provoked; observe that, since the game ends upon **win**, if $\mathcal{E}$ occurs, it occurs before condition **win** in one of the *inject* oracles is triggered.

It suffices to upper bound the probability of $\mathcal{E}$ occurring in, say, H_1. This is achieved via a reduction: Consider an attacker $\mathcal{B}_1$ against correctness that uses $\mathcal{A}$ and attempts to simulate H_1 to $\mathcal{A}$. The simulation proceeds as follows (the oracles $\mathcal{B}_1$ interacts with are referred to as $\mathcal{B}_1$'s *own* oracles):

- Initially, choose a bit $b \in \{0, 1\}$ randomly and initialize empty chall. Run $\mathcal{A}$.
- Simulate SM game oracles to $\mathcal{A}$ as follows (oracles corresponding to B are simulated similarly):
  - **transmit-A**: simply forward (to own **transmit-A** and back).
  - **chall-A**(m_0, m_1, r): from $\mathcal{A}$'s previous queries, determine whether $\mathsf{safe\text{-}ch}_{\mathsf{A}}$ will be satisfied after **ep-mgmt**. If so, call own **transmit-A**(m_b, r), obtain and record answer c in $\mathsf{chall} \xleftarrow{+} (\mathsf{A}, c)$, and pass c to $\mathcal{A}$.
  - **deliver-A**(c): call **deliver-A**(c) and pass answer to $\mathcal{A}$ except when $(\mathsf{B}, c) \in \mathsf{chall}$, then replace m' by $\perp$ in the answer and remove triple from chall.
  - **inject-A**(c): simply forward.

- **corr-A**: if $\mathsf{B} \notin \mathsf{chall}$, call own **corr-A** and forward response to $\mathcal{A}$.

The oracles corresponding to B are defined analogously.

By inspection, one observes that $\mathcal{B}_1$ correctly, i.e., identically to H_1, handles challenges and injections in the game. In particular, since the state of a party remains unchanged if Rcv rejects a ciphertext (cf. Property (A) in Definition 7), the reduced inject oracle of the correctness game is identical to the actual inject oracle as long as the attacker does not provoke **win** inside an injection oracle. The latter, however, would preclude $\mathcal{E}$ from occurring. Hence, the probability that $\mathcal{B}_1$ wins the correctness game is equal to the probability that $\mathcal{A}$ provokes $\mathcal{E}$ in H_1. □

A.2 Getting Rid of Injections

Hybrid H_2. Hyrbid H_2 works as H_1, but with *reduced* inject oracles, as defined in Section 3.3. In the following, denote the advantage of $\mathcal{A}$ in H_2 by $\mathrm{Adv}^{\mathsf{SM}}_{H_1}(\mathcal{A})$.

Lemma 17. *There exists a (t', q, q_{ep})-attacker $\mathcal{B}_2$ with $t' \approx t$ such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_1}(\mathcal{A}) \;\leq\; \mathrm{Adv}^{\mathsf{SM}}_{H_2}(\mathcal{A}) + q_{\mathsf{ep}}^2 \cdot \mathrm{Adv}^{\mathsf{SM}}_{\mathrm{auth},\Delta_{\mathsf{SM}}}(\mathcal{B}_2) \;.$$

Proof. In order to distinguish H_1 and H_2, $\mathcal{A}$ must provoke the **win** condition in the inject oracles since H_1 and H_2 behave identically otherwise. Therefore, it suffices to upper bound the probability of this event $\mathcal{E}$ in, say, H_2, which is achieved via a reduction: Consider an attacker $\mathcal{B}_2$ against authenticity that uses $\mathcal{A}$ and attempts to simulate H_2 to $\mathcal{A}$. The simulation proceeds as follows (the oracles $\mathcal{B}_2$ interacts with are referred to as $\mathcal{B}_2$'s *own* oracles):

- Initially, choose values $t^* \in [q_{\mathsf{ep}}]$ and $t^*_{\mathsf{L}} \in [t^* - \Delta_{\mathsf{SM}}]$ uniformly at random and call own **init**(t^*_{L}, t^*); choose bit $b \in \{0,1\}$ randomly and initialize empty chall. Run $\mathcal{A}$.
- Simulate SM game oracles to $\mathcal{A}$ as follows (oracles corresponding to B are simulated similarly):
  - **transmit-A**: simply forward (to own **transmit-A** and back).
  - **chall-A**(m_0, m_1, r): from $\mathcal{A}$'s previous queries, determine whether $\mathsf{safe\text{-}ch_A}$ will be satisfied after **ep-mgmt**. If so, call own **transmit-A**(m_b, r), obtain and record answer c in $\mathsf{chall} \xleftarrow{+} (\mathsf{A}, c)$, and pass c to $\mathcal{A}$.
  - **deliver-A**(c): call **deliver-A**(c) and pass answer to $\mathcal{A}$ except when $(\mathsf{B}, c) \in \mathsf{chall}$, then replace m' by $\perp$ in the answer and remove triple from chall.
  - **inject-A**(c): simply forward.
  - **corr-A**: if $\mathsf{B} \notin \mathsf{chall}$, call own **corr-A** and forward response to $\mathcal{A}$.

The oracles corresponding to B are defined analogously.

For the analysis of the reduction, one must argue that the probability that $\mathcal{B}_2$ wins the authenticity game is at least $q_{\mathsf{ep}}^{-2} \cdot \mathsf{P}[\mathcal{E}]$ (where the probability is in H_2). To that end, observe that $\mathcal{B}_2$ tries to guess

- the epoch t^* of the *first* successful injection query $\mathcal{A}$ makes, and
- the time t^*_{L} of the last corruption event before the healing event that should have protected against the injection.

Assume that t^* is *even*, i.e., in the epoch under attack, B is the sender and A the receiver; the case where t^* is odd works analogously.

It remains to argue that whenever the guess is correct, the simulation succeeds. To that end, observe first that, by virtue of Property (E) in Definition 7, $t_{\mathsf{A}} \geq t^* - 1$ when the successful injection occurs. Hence, no successful inject query can be made before A reaches epoch $t^* - 1$. Moreover, Properties (C) and (D) imply that the **win**-event cannot be the result of a replay attack or by having A receive a ciphertext she sent.

Observe that the attacker correctly handles challenges and injections. In particular, since the state of a party remains unchanged if Rcv rejects a ciphertext (cf. Property (A) in Definition 7), the reduced inject oracle is identical to the actual inject oracle as long as the attacker does not provoke **win** inside an injection oracle.

Moreover, if the guess is correct, $\mathcal{B}_2$ cannot lose the authenticity game due to the condition on t_{L}. Finally, $\mathcal{B}_2$ loses due to the corruption condition on $\mathcal{A}$ only if $\mathcal{A}$ asks to corrupt A (the receiver in epoch t^*) when $t_{\mathsf{A}} > t^*_{\mathsf{L}}$, which would either contradict the correctness of the guess, or it occurs when $t_{\mathsf{A}} \geq t^*$, in which case all epoch-t^* messages become compromised and **win** could not be provoked by $\mathcal{A}$ anymore anyway. Finally, the view of $\mathcal{A}$ is independent of (t^*_{L}, t^*) (at least until $\mathcal{A}$ wins or loses), and hence the probability of a correct guess is at least q_{ep}^{-2}. □

A.3 Substituting Random Encryptions

Hybrids $H_{3,i}$. These hybrids, for $i = 0, 1, \ldots, q_{\mathsf{ep}}$, work as H_2, but challenges up to epoch i are created by encrypting a fixed message $\bar{m}$ instead of m_b. Observe that $H_{3,0}$ is identical to H_2. Denote the advantage of $\mathcal{A}$ by $\mathrm{Adv}^{\mathsf{SM}}_{H_{3,i}}(\mathcal{A})$.

Lemma 18. *For each $i = 1, \ldots, q_{\mathsf{ep}}$, there exists an attacker $\mathcal{B}_{3,i}$ (comparable in efficiency to $\mathcal{A}$) such that*

$$\mathrm{Adv}^{\mathsf{SM}}_{H_{3,i-1}}(\mathcal{A}) \;\leq\; \mathrm{Adv}^{\mathsf{SM}}_{H_{3,i}}(\mathcal{A}) + i \cdot \mathrm{Adv}^{\mathsf{SM}}_{\mathrm{priv},\Delta_{\mathsf{SM}}}(\mathcal{B}_{3,i}) \; .$$

Proof. Consider an attacker $\mathcal{A}$ against SM security and a reduction attacker $\mathcal{B}_{3,i}$ (against privacy) that works as follows (the oracles $\mathcal{B}_{3,i}$ interacts with are referred to as $\mathcal{B}_{3,i}$'s *own* oracles):

- Initially, let $t^* = i$ and choose value $t^*_{\mathsf{L}} \in [i]$ uniformly at random and call own **init**(t^*_{L}, t^*); choose bit $b \in \{0,1\}$ randomly and initialize empty chall. Run $\mathcal{A}$.
- Simulate SM game oracles to $\mathcal{A}$ as follows (oracles corresponding to B are simulated similarly):
  - **transmit-A**: simply forward (to own **transmit-A** and back).
  - **chall-A**(m_0, m_1, r):
    - First $i - 1$ epochs: from $\mathcal{A}$ previous queries, determine whether $\mathsf{safe\text{-}ch_A}$ will be satisfied after **ep-mgmt**. If so, call own **transmit-A**$(\bar{m}, r)$, obtain and record answer c in $\mathsf{chall} \xleftarrow{+} (\mathsf{A}, c)$, and pass c to $\mathcal{A}$.
    - Epoch i (for **chall-A** if i is odd and **chall-B** if i is even): pass $(\bar{m}, m_b, r)$ to own **chall-A**; obtain and pass answer c to $\mathcal{A}$.
    - Epochs after ith: from $\mathcal{A}$ previous queries, determine whether $\mathsf{safe\text{-}ch_A}$ will be satisfied after **ep-mgmt**. If so, call own **transmit-A**(m_b, r), obtain and record answer c in $\mathsf{chall} \xleftarrow{+} (\mathsf{A}, c)$ and pass c to $\mathcal{A}$.

– **deliver-A**(c): call **deliver-A**(c) and pass answer to $\mathcal{A}$ except when $(\mathsf{B}, c) \in \mathsf{chall}$, then replace m' by $\perp$ in the answer and remove triple from chall.

– **inject-A** simply forward.

– **corr-A**: If $\mathsf{B} \notin \mathsf{chall}$, call own **corr-A** and forward response to $\mathcal{A}$.

The oracles corresponding to B are defined analogously.

Observe that $\mathcal{B}_{3,i}$ tries to guess the time t^*_{L} of the last corruption event before epoch i. *If* the guess then $\mathcal{B}_{3,i}$ simulates $H_{3,i-1}$ to $\mathcal{A}$ if the secret bit inside the privacy game is 0 and $H_{3,i}$ if it the bit is 1. In particular, observe that (if the guess is correct) $\mathcal{B}_{3,i}$ loses the privacy game only if $\mathcal{A}$ would lose the SM game (by an argument similar to that in the reduction to authenticity in Lemma 17). In particular, the guess t^*_{L} is independent of $\mathcal{A}$'s view (at least until $\mathcal{A}$ loses). □

Clearly, for $H_{3,q_{\mathsf{ep}}}$, $\mathrm{Adv}^{\mathsf{SM}}_{H_{3,q_{\mathsf{ep}}}}(\mathcal{A}) = 0$. Theorem 1 now follows by combining Lemmas 16 to 18.

B Additional Preliminaries

B.1 Public-Key Encryption

Definition 18. *A* public-key encryption (PKE) scheme *consists of three algorithms* $\mathsf{PKE} = (\mathsf{PKEG}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

- ***Key generation:*** PKEG *takes a (implicit) security parameter and outputs a fresh key pair* $(\mathsf{ek}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{PKEG}$*.*

- ***Encryption:*** Enc *takes a public key* ek *as well as a message* m *and produces a ciphertext* $e \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{ek}, m)$*.*

- ***Decryption:*** Dec *takes a secret key* sk *and a ciphertext* e *and outputs a message* $m \leftarrow \mathsf{Dec}(\mathsf{sk}, e)$*.*

Correctness. A PKE must satisfy the following standard correctness property:

$$\mathsf{P}[(\mathsf{ek}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{PKEG}, c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{ek}, m), m' \leftarrow \mathsf{Dec}(\mathsf{sk}, c) : m' = m] \;=\; 1\ .$$

Security. This work considers CCA security, which asks that no adversary be able to distinguish encryptions of two different messages even given access to a decryption oracle. This is captured by considering the following game: It chooses a random bit $b \leftarrow \{0,1\}$, produces a key pair $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{PKEG}$ and passes ek to the attacker $\mathcal{A}$. The attacker specifies two messages m_0 and m_1 and is then given $c^* \leftarrow \mathsf{Enc}(\mathsf{ek}, m_b)$. Moreover, $\mathcal{A}$ is given access to a decryption oracle for sk, but it is not allowed to query the challenge ciphertext c^*. The attacker $\mathcal{A}$ wins the game if it correctly guesses b. The advantage of $\mathcal{A}$ is denoted by $\mathrm{Adv}^{\mathsf{PKE}}_{\mathrm{CCA}}(\mathcal{A})$; the attacker is parametrized by its running time t.

Definition 19. *A public-key encryption scheme* Π *is* (t, ε)-CCA-secure *if for all t-attackers $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathsf{PKE}}_{\mathrm{CCA}}(\mathcal{A}) \;\leq\; \varepsilon\ .$$

B.2 Digital Signatures

Definition 20. *A* digital-signature scheme (DSS) *is a triple of algorithms* $\mathsf{DSS} = (\mathsf{DSG}, \mathsf{Sign}, \mathsf{Verify})$ *with the following syntax:*

- ***Key Generation:*** *Given an (implicit) security parameter, key generation algorithm* Gen *derives a key pair* $(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{DSG}$.
- ***Signing:*** *Given a message* m *and a signing key* sk*, the signing algorithm* Sgn *produces a signature* $s \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$.
- ***Verification:*** *Given a message* m*, a signature* s*, and a verification key* vk*, the verification algorithm outputs a decision bit* $d \leftarrow \mathsf{Verify}(\mathsf{vk}, m, s)$.

A DSS is called deterministic *if* Sgn *is deterministic.*

Digital signature schemes in this work are required to be deterministic. Note that any DSS can be converted into a deterministic one by (1) having the signing key include the key to a PRF and (2) generating the coins needed to sign a message m by evaluating the PRF on input m.

Correctness. We require standard correctness property that signatures generated under a signing key always pass verification under the corresponding verification key:

$$\forall m : \quad \mathsf{P}[(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{DSG}, s \leftarrow \mathsf{Sign}(\mathsf{sk}, m) : 1 = \mathsf{Verify}(\mathsf{vk}, m, s)] = 1 \ .$$

Security. DS schemes in this work must satisfy the (standard) notion of *existential unforgeability under chosen-message attacks* (EUF-CMA). This is captured by the following game: It generates a key pair $(\mathsf{sk}, \mathsf{vk})$ and passes vk to the attacker. Then, the attacker is given access to a signing oracle for sk. The adversary wins the game if it is able to eventually output a *forgery*, i.e., a pair (m, s) such that m has not been queried to the signature oracle, but s is a valid signature for m under vk. The advantage of the attacker $\mathcal{A}$ against a scheme DSS is denoted by $\mathrm{Adv}^{\mathsf{DSS}}_{\mathrm{CMA}}(\mathcal{A})$.

C Key-Encapsulation Mechanisms

C.1 ElGamal KEM

In the (standard) ElGamal KEM, a public key has the form $h = g^s$, where g is the generator of cyclic group G of prime order q and where the secret key $s \in \mathbb{Z}_q$ is chosen uniformly at random. That is,

$$(h, s) \ = \ (g^s, s) \ \leftarrow\!\!\$ \ \ \mathsf{Gen} \ .$$

To encapsulate a key, let $r \in \mathbb{Z}_q$ be uniformly random and

$$(c, k) \ = \ (g^r, h^r) \ \leftarrow\!\!\$ \ \ \mathsf{Enc}(h) \ .$$

Finally, for decapsulation,

$$k \ = \ c^s \ \leftarrow \ \mathsf{Dec}(s, c) \ .$$

It is well known and easy to see that the security of this scheme follows immediately form the DDH assumption.

C.2 Frodo KEM

This section *briefly* recapitulates the Frodo KEM; for more details the reder is referred to the original paper [6]. The Frodo KEM $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is based on the learning with errors (LWE) problem. It is parametrized by a size parameters n and $\bar{n}$, a modulus q, and an error distribution χ over $\mathbb{Z}_q$. The key generation algorithm Gen chooses a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ uniformly at random, and it samples matrices $\mathbf{S}, \mathbf{E} \leftarrow \chi(\mathbb{Z}_q^{n \times \bar{n}})$ independently, where $\chi(\mathbb{Z}_q^{n \times \bar{n}})$ is the distribution on matrices obtained by sampling each entry independently according to χ. The public key is $\mathsf{pk} = (\mathbf{A}, \mathbf{B} := \mathbf{AS} + \mathbf{E})$, and the secret key is $\mathsf{sk} = \mathbf{S}$.

In order to encapsulate a key for pk, algorithm Enc chooses $\mathbf{S}', \mathbf{E}' \sim \chi(\mathbb{Z}_q^{\bar{n} \times n})$, and $\tilde{\mathbf{E}}' \sim \chi(\mathbb{Z}_q^{\bar{n} \times \bar{n}})$ independently and computes

$$\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}' \qquad \text{as well as} \qquad \mathbf{V}' \leftarrow \mathbf{S}'\mathbf{B} + \tilde{\mathbf{E}}' \; .$$

Then, from $\mathbf{V}'$, reconciliation information $\mathbf{C}' \leftarrow \langle \mathbf{V}' \rangle_{2^B}$ as well as the key $K \leftarrow \lfloor \mathbf{V}' \rceil_{2^B}$ are computed (see below). The ciphertext is $c = (\mathbf{B}', \mathbf{C}')$.

To decapsulate the key from a ciphertext c with secret key $\mathsf{sk} = \mathbf{S}$, algorithm Dec computes $K \leftarrow \mathsf{rec}(\mathbf{B}'\mathbf{S}, \mathbf{C}')$, where rec is a function that uses the reconciliation information to recover the key.

Reconciliation. Assume for simplicity that q is a power of two, and let B and $\bar{B}$ such that $B + \bar{B} = \log q$. Observe that both the the decrypting party computes an approximation $\mathbf{B}'\mathbf{S}$ of matrix $\mathbf{V}'$. In order to extract B random-looking bits from each entry v of matrix $\mathbf{V}'$, the encrypting party applies the function

$$\lfloor \cdot \rceil_{2^B} : v \mapsto \left\lfloor 2^{-\bar{B}} v \right\rceil \mod B$$

to v. This function can be seen to split $\mathbb{Z}_q$ into B intervals of length $\bar{B}$ centered around 0. Since the decrypting party can only compute an approximation of each entry v, the following reconciliation information

$$\langle \cdot \rangle_{2^B} : v \mapsto \left\lfloor 2^{-\bar{B}+1} v \right\rfloor \mod 2$$

is passed to the decrypting party. This function can be seen to partition each interval above into two parts, according the $(\bar{B}+1)^{\text{st}}$ most significant bit. The reconciliation information helps the reconciliation function rec in cases where the value v is close to an interval border (as long as the error is not larger than $2^{\bar{B}-2}$) and can be shown not to reveal information about $\lfloor v \rceil_{2^B}$. For further information, see [6].