# A Unified and Composable Take on Ratcheting

Daniel Jost(✉), Ueli Maurer, and Marta Mularczyk

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
{dajost,maurer,mumarta}@inf.ethz.ch

**Abstract.** Ratcheting, an umbrella term for certain techniques for achieving secure messaging with strong guarantees, has spurred much interest in the cryptographic community, with several novel protocols proposed as of lately. Most of them are composed from several sub-protocols, often sharing similar ideas across different protocols. Thus, one could hope to reuse the sub-protocols to build new protocols achieving different security, efficiency, and usability trade-offs. This is especially desirable in view of the community's current aim for group messaging, which has a significantly larger design space. However, the underlying ideas are usually not made explicit, but rather implicitly encoded in a (fairly complex) security game, primarily targeted at the overall security proof. This not only hinders modular protocol design, but also makes the suitability of a protocol for a particular application difficult to assess.

In this work we demonstrate that ratcheting components can be modeled in a composable framework, allowing for their reuse in a modular fashion. To this end, we first propose an extension of the Constructive Cryptography framework by so-called global event histories, to allow for a clean modularization even if the component modules are not fully independent but actually subtly intertwined, as in most ratcheting protocols. Second, we model a unified, flexibly instantiable type of strong security statement for secure messaging within that framework. Third, we show that one can phrase strong guarantees for a number of sub-protocols from the existing literature in this model with only minor modifications, slightly stronger assumptions, and reasonably intuitive formalizations.

When expressing existing protocols' guarantees in a simulation-based framework, one has to address the so-called commitment problem. We do so by reflecting the removal of access to certain oracles under specific conditions, appearing in game-based security definitions, in the real world of our composable statements. We also propose a novel non-committing protocol for settings where the number of messages a party can send before receiving a reply is bounded.

M. Mularczyk—Research supported by the Zurich Information Security and Privacy Center (ZISC).


D. Hofheinz and A. Rosen (Eds.): TCC 2019, LNCS 11892, pp. 180–210, 2019.
https://doi.org/10.1007/978-3-030-36033-7_7

# 1 Introduction

## 1.1 Secure Messaging and Ratcheting

Secure-messaging (SM) protocols attempt to provide strong security guarantees to two parties that communicate over an asynchronous network. Apart from protecting confidentiality and integrity of messages, the desired properties include forward secrecy and healing from a state or randomness exposure. The latter properties are addressed by the so-called ratcheting protocols, by having the parties continuously update their secret keys.

The term ratcheting on its own does not carry any formal meaning; rather, it is an umbrella term for a number of different guarantees, somehow related to the concept of updating keys. One notable example of ratcheting is the widely-used Signal protocol [21] with its double-ratchet algorithm, formally analyzed in [1,7]. Furthermore, there exist protocols with much stronger guarantees, but that require the messages to be delivered in order [8–10,24]. Protocols with the stronger guarantee of immediate out-of-order decryption have been proposed in [1]. While the majority of the literature considers secure communication, some works view ratcheting as a property of key exchange instead [2,24].

A number of proposed protocols pursue similar goals, but each achieves a slightly different trade-off between security, efficiency and usability. Moreover, each construction comes with its own—usually fairly complex—security game, intermediate abstractions, and primitives. This renders them hard to compare and hinders achieving new trade-offs that would result from combining ideas from different protocols. This motivates the goal of this work, which is to facilitate a systematic, modular and composable analysis of secure-messaging protocols.

## 1.2 Composable Security

While a game-based systematization of secure messaging could certainly address some of the aforementioned concerns, composable frameworks, such as [4,14,18,23], provide some distinct advantages.

First, security under (universal) composition is a stronger notion: the guarantees are provided even if a protocol is executed in an arbitrary environment, alongside other protocols. So far, no SM protocol provably achieves this (in fact, even the weaker notion of security under parallel self composition has not been analyzed). Moreover, composable frameworks facilitate modularity. One can define components with clean abstraction boundaries (e.g., a secure channel) and use their idealized versions in a higher-level protocol (and its proof). The overall security of the composed protocol follows from the composition theorem. This stands in contrast with game-based definitions, where security of the components and the overall protocol is expressed by a number of games, and one has to show that winning the security game for the overall protocol implies, via reductions, winning one security game for a component. Finally, guarantees expressed in a composable framework usually have more evident semantics,

obtained from directly considering how a protocol is used, rather than a hypothetical interaction of an adversary with a simplified game that encodes excluded attacks.

Unfortunately, secure messaging does not render itself easily to a modular, composable analysis. One reason for this is the difficulty in drawing the right abstraction boundaries. Roughly, the guarantees for a channel heavily depend on other components in the system, for example, we may want to say that the confidentiality of a message is protected only if some memory contents do not leak. This problem also appears in the analysis of some protocols from different contexts (e.g. TLS [12]), which often violate the rules of modularity.

Furthermore, we encounter the so-called "commitment problem" of simulation-based security definitions. Intuitively, the natural composable guarantees are too strong and provide additional security that seems to carry little significance in practice, and that can only be achieved with (stronger) setup assumptions and at an efficiency loss. To address this problem, a number of approaches have been proposed—none of them, however, being able to fully satisfactorily formalize the weaker guarantees achieved by regular schemes. First, the notion of non-information oracles [6] has been proposed that essentially embeds a game-base definition in a composable abstraction module. Second, a line of work considers stronger, i.e., super-polynomial, simulators [3,22,25]. Protocols in those models, however, still have to rely on additional setup and special primitives.

### 1.3 Contributions

This paper makes both conceptual and technical contributions. Conceptual contributions to composable security frameworks are the notion of global event histories as well as a modeling technique for circumventing the so-called commitment problem. Technical contributions include the modeling of ratcheting (sub-)protocols in a composable framework as well as a novel protocol that achieves adaptive security, i.e., the strongest form of composable security, under certain restrictions.

*Global Event Histories.* Composable frameworks are based around the idea of independent modules (e.g. channels, keys, or memory resources) that are constructed by one protocol and then used by another protocol in the next construction step. However, in many settings, in particular as they occur in modeling ratcheting protocols, these components are subtly correlated which seems to violate modularity. For example, a channel (one module) can become insecure when a key (another, apparently independent module) is leaked to the adversary.

We address this problem by two conceptual ideas. First, we parameterize resources by several (discrete) parameters—which can be thought of as a switch with two or more positions—which can downgrade the security of a resource, e.g. switch a channel from non-leakable (i.e. confidential) to leakable. Second, we introduce the notion of *global event histories* defined for the entire real (or ideal) world, where a history is a list of events having happened at a module (e.g.

a message being input by Alice or a message having leaked to the adversary). A key idea is now that the switch settings of the modules can be defined by predicates (or, more generally, multi-valued functions) of the global event history. This allows us to draw meaningful abstraction boundaries for secure messaging, but we believe that the concept of event histories is of independent interest and may enable modular analyses for settings where this was previously difficult.

Formally, we use the Constructive Cryptography (CC) framework [15,18], and in particular a slight modification of its standard instantiation to model the event history. Since the composition theorem of CC is proved on an abstract level, we do not need to re-prove it.

*Expressing the Guarantees Provided by Ratcheting.* Our goal is to capture the guarantees provided by ratcheting (sub-)protocols in a general fashion to make them reusable in different protocols or contexts. This is in contrast to existing game-based definitions, which usually formalize exactly what is required by the next sub-protocol for the overall protocol's security proof to go through.

This goal is achieved by considering parameterized resources as described above and modeling the goal of a (sub-)protocol as improving a certain parameter while leaving the other parameters unchanged, independently of what they are. One can think of a protocol improving certain switch positions (e.g. making a channel confidential), independently of the other switch positions.

In this paper, we consider three ratcheting sub-protocols. We start with a simple authentication protocol in the unidirectional setting which constantly updates keys. As a more involved example, we consider the use of hierarchical identity-based encryption to provide confidentiality. As a third example, we analyze continuous key agreement, a notion introduced by Alwen et al. [1] to abstract the asymmetric ratcheting layer of Signal. On the way, we discover cases where the existing game-based notions are insufficient to prove the stronger, more modular, statements that don't fix the properties (i.e., the switch positions) of the assumed network, but where they can be achieved by simple modifications.

*Solutions to the Commitment Problem.* When modeling ratcheting protocols, we encounter the so-called commitment problem: the simulator would have to output a simulated value (e.g. a ciphertext) which at a later stage must be compatible with another value (e.g. a leaked key) not initially known to him. Since this is generally impossible, we address this problem in two alternative ways.

On one hand, we propose a technique that allows to transform many standard SM protocols into protocols that achieve full composable security, at the expense of an efficiency lost, as well as being restricted to only sending a bounded number of messages before receiving a reply from the other party. We apply this technique to the HIBE protocol mentioned above and construct its fully composable version.

On the other hand, we can retain composable statements of regular protocols by restricting the adversary's capabilities in the real Roughly, we observe that game-based definitions do not encounter the commitment problem, because they

disable certain sequences of oracle calls. For example, if the adversary calls the challenge oracle to obtain a ciphertext, she cannot immediately call the expose oracle that returns the secret key, since this would allow her to trivially win. We give a composable semantic to such conditions by making some real-world components secure by assumption after certain sequences of events. For example, after a message is sent, the memory storing the secret key becomes secure.

### 1.4 Outline

In Sect. 3 we extend Constructive Cryptography to include the global event history. Based on this, in Sect. 4, we introduce a simple and generic type of security statement for SM protocols. (In the full version [11] we extend it to encompass ratcheting as a key-exchange primitive.) In Sect. 5, we demonstrate how the security guarantees of ratcheting components can be phrased in this model. In Sect. 6, we introduce a novel non-committing ratcheting protocol that achieves full simulation-based security for a bounded number of messages.

## 2 Preliminaries: Constructive Cryptography

### 2.1 The Real-World/Ideal-World Paradigm

Many security definitions, and in particular most composable security frameworks [4,14,18,23], are based on the real-world/ideal-world paradigm. The real world models the use of a protocol, whereas the ideal world formalizes the security guarantees that this protocol is supposed to achieve.

The security statement then affirms that the real word is "just-as-good" as the ideal world, meaning that for all parties, no matter whether honest or adversarial, it does not make a difference whether they live in the real or ideal world. Hence, if the honest parties are content with the guarantees they get in the ideal world, they can safely execute the protocol in the real world instead.

### 2.2 Resources

In each composable framework there is some notion of a module that exports a well-defined interface in a black-box manner to the rest of the world. In the UC framework such a module is called a *functionality*. In the Constructive Cryptography (CC) framework [15,18] such a module is called a *resource*. One of the main differences is that in CC a world consists entirely of resources and the environment (called a distinguisher). So while UC distinguishes between the real world, where the parties can only send messages to each other, and a hybrid world, where they additionally access some ideal functionalities, in CC everything, including communication, is a resource. For example, a security statement about two parties using authenticated encryption to transmit a message is phrased as a real world containing two resources—an insecure channel

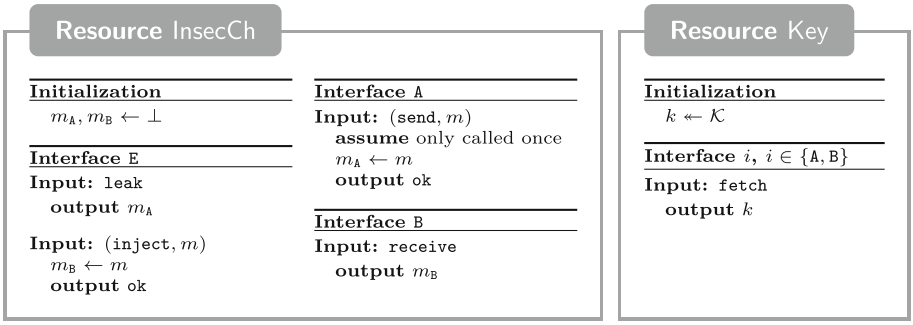


**Fig. 1.** The assumed real-world resources of the authenticated-encryption example: an insecure channel and a shared key. The insecure channel exports three interface `A`, `B`, and `E`, understood to be controlled by the respective parties Alice, Bob, and Eve, whereas the key resource only exports two interfaces.

and a shared key—which are then used by the protocol to construct the ideal world consisting of a secure channel. See Fig. 1 for a description of the real-world resources.

A resource is a reactive system that allows interaction at one or several *interfaces*, i.e. upon providing an input at one of the interfaces, the system provides an output. In this work, we only consider systems where the output is produced at the same interface the input was given. Formally, resources are modeled as random systems [16], where the interface address is encoded as part of the inputs. However, a reader unfamiliar with CC may simply think of a resource with $n$ interfaces as $n$ oracles that share a joint state. Note that there is no formal notion of a party in constructive cryptography; they only give meaning to the construction statements, by thinking of each interface being controlled by some party. Since in this work we make statements about messaging between two honest parties, called Alice and Bob, in the presence of a global adversary, called Eve, we usually label the interfaces accordingly, indicating how the assignment of interfaces to parties should be understood.

A set of resources can be composed into a single one. The interface set of the composed resource corresponds to the union of the ones from the composed resources. Returning to our example of authenticated encryption, in the real world we have both an insecure channel $\mathsf{InsecCh}$ and a key $\mathsf{Key}$, where the former has three interfaces and the latter two. The composed resource, denoted $[\mathsf{InsecCh}, \mathsf{Key}]$, is a resource with five interfaces, each of them addressed by a tuple consisting of the resource's name and the interface's original name.

We describe our resources using pseudo-code (c.f. Fig. 1). The following conventions are followed: each resource has an initialization procedure initializing all the persistent variables (all other variables are understood to be volatile). Formally this initialization is called upon invoking any arbitrary interface for the first time. Each interface exposes one or more capabilities, each of them described by a keyword (e.g. `send` in case of a channel), and the (potential empty) list of arguments (e.g., $m$). Furthermore, we use the **assume** command,

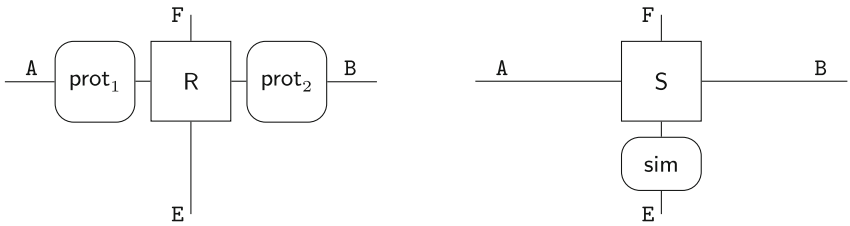


**Fig. 2.** Execution of the protocol in the real world by Alice and Bob (left) and the ideal world with the simulator attached to Eve's interface (right). The free interface on the top is accessed directly by the environment in both worlds.

which should be understood as a shortcut for explicitly tracking the respective condition and returning an error symbol $\perp$ in case the condition is violated. In Fig. 1, the keyword **assume** is used the specify that the channel is single-use.

### 2.3 Converters

The protocol execution in CC is modeled by *converters*, each of which expresses the local computation executed by one party. (The name converter derives from the property that a converter attached to a resource converts it into another "ideal" one.) A converter expects to be connected to a given set of interfaces at the "inside", and emulates a certain set of interfaces at the "outside". Upon an input at one of the emulated interfaces, the converter is allowed to make a bounded number of oracle queries to the inside interfaces (recall that a resource always returns at the same interface it was queried), before returning a value at the same emulated interface. For a converter $\mathsf{prot}$ and a resource $\mathsf{R}$, we denote by $\mathsf{R}' := \mathsf{prot}^{\{I_1,\ldots,I_n\}}\mathsf{R}$ the resource obtained from connecting the converter to the subset $\{I_1, \ldots, I_n\}$ of the interfaces. The resource $\mathsf{R}'$ no longer exposes those interfaces to the world, but the ones emulated by $\mathsf{prot}$ instead. We usually omit specifying the set $\{I_1, \ldots, I_n\}$ and just write for instance $\mathsf{prot}^{\mathsf{A}}\mathsf{R}$, denoting that it is connected to all of Alice's interfaces.

### 2.4 The Construction Notion

Security is then defined following the real-world/ideal-world paradigm, stating that in every environment the real world should behave the same way as the ideal one. The real world, as depicted in Fig. 2, thereby consists of the assumed resource $\mathsf{R}$ to which the converters are attached, each to a subset of the respective party's interfaces. The ideal world, on the other hand, consists of the constructed resource $\mathsf{S}$ with a simulator (which is a converter) attached to Eve's interfaces.

Behaving the same way is formalized using the notion of a distinguisher, that can make oracle queries to the resource's interfaces and then outputs a

bit, indicating whether it believes to interact with the real or ideal world. More formally, in the special case of two honest parties Alice and Bob and a global adversary Eve, the goal of a distinguisher $\mathbf{D}$ is to distinguish the real world $\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R}$ from the ideal world $\mathsf{sim}^{\mathtt{E}}\,\mathsf{S}$. The advantage of $\mathbf{D}$ is defined as

$$\Delta^{\mathbf{D}}\left(\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R},\ \mathsf{sim}^{\mathtt{E}}\,\mathsf{S}\right) \coloneqq \Pr\left[\mathbf{D}(\mathsf{sim}^{\mathtt{E}}\,\mathsf{S}) = 1\right] - \Pr\left[\mathbf{D}(\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R}) = 1\right].$$

Let $\epsilon$ denote a function mapping distinguishers to values in $[-1, 1]$. Then, the protocol $(\mathsf{prot}_1, \mathsf{prot}_2)$, when attached to $\mathtt{A}$ and $\mathtt{B}$, is said to *construct* $\mathsf{S}$ from $\mathsf{R}$ within $\epsilon$, and with respect to $\mathsf{sim}$ attached to $\mathtt{E}$, if

$$\forall \mathbf{D} : \Delta^{\mathbf{D}}\left(\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R},\ \mathsf{sim}^{\mathtt{E}}\,\mathsf{S}\right) \leq \epsilon(\mathbf{D}).$$

Note that we require the sets of interfaces controlled by Alice, Bob, and Eve, respectively, to be pairwise disjoint. They however do not have to completely partition the set of interfaces. The remaining interfaces are called *free interfaces* to which the distinguisher has direct access in both worlds.

For simplicity, in this work we consider an asymptotic setting only (although we usually do not make the asymptotics explicit) where all resources and converters are assumed to be efficiently implementable. We then write

$$\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R} \approx \mathsf{sim}^{\mathtt{E}}\,\mathsf{S},$$

if $\Delta^{\mathbf{D}}\left(\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R},\ \mathsf{sim}^{\mathtt{E}}\,\mathsf{S}\right)$ is negligible for every efficient distinguisher $\mathbf{D}$, and simply say that $(\mathsf{prot}_1, \mathsf{prot}_2)$ constructs $\mathsf{S}$ from $\mathsf{R}$ if there exists an efficient simulator $\mathsf{sim}$ achieving this.

Note that the notion of construction is analogous to the notion of secure realization in the UC framework. In contrast to UC, however, the set of all resource instances within a construction statement is fixed. The distinguisher does not instantiate resources or protocols, or assign session identifiers. Dynamic availability properties of resources can obviously still be modeled as part of the resources themselves, though.

### 2.5 Composition

The notion of construction is composable, which intuitively means that if a protocol $(\mathsf{prot}_1, \mathsf{prot}_2)$ constructs $\mathsf{S}$ from $\mathsf{R}$, and another protocol $(\mathsf{prot}_1', \mathsf{prot}_2')$ constructs $\mathsf{T}$ from $\mathsf{S}$, then the combined protocol constructs $\mathsf{T}$ from $\mathsf{R}$. This is known as *sequential composition*. Additionally, if $(\mathsf{prot}_1, \mathsf{prot}_2)$ constructs $[\mathsf{S}_1, \ldots, \mathsf{S}_i]$ from $[\mathsf{R}_1, \ldots, \mathsf{R}_j]$, for some $i$ and $j$, then for every set of (efficiently implementable) resources $\{\mathsf{T}_1, \ldots, \mathsf{T}_n\}$ it also holds that $(\mathsf{prot}_1, \mathsf{prot}_2)$ constructs $[\mathsf{S}_1, \ldots, \mathsf{S}_i, \mathsf{T}_1, \ldots, \mathsf{T}_n]$ from $[\mathsf{R}_1, \ldots, \mathsf{R}_j, \mathsf{T}_1, \ldots, \mathsf{T}_n]$, where the interfaces of the additional resources $\mathsf{T}_1, \ldots, \mathsf{T}_n$ are treated as free in the construction. This property is known as *parallel composition*.

Both properties are proven in [18,19] for a more abstract notion of resources being "just-as-good", of which the here introduced indistinguishability notion is a special case. Together, the two properties form the equivalent to the universal composability property of the UC framework.

## 3 Constructive Cryptography with Events

In this section we generalize the Constructive Cryptography framework to allow for better modularization. More specifically, we introduce another instantiation of resources and the "just-as-good" notion, thereby inheriting the composition theorem of CC that is proven on a more abstract level.

**Motivation.** Recall that SM protocols are difficult to modularize, because the guarantees for a given message depend on the dynamically changing state of other components in the system, such as whether the state leaked or the adversary tampered with a previous message. In traditional CC, where the abstraction boundary of a resource is just the input-output behavior, properly accounting for those dependencies would essentially force us to model the whole SM application as monolithic resource. In this section, we therefore extend the notions of resources and construction to relax the abstraction boundary in a clean and well-controlled manner, which will allow for such dependencies between different resources. More concretely, we introduce a global event history. Each resource is then allowed to trigger events from a predefined set (e.g. indicating that a party's state leaked), on which the behavior of other resources can then depend. The event history is visible to the environment, the resources, and the simulator.[1]

**The Global Event History.** We model events as a generalization of monotone binary outputs (MBO) introduced by Maurer et al. [17]. Roughly, an MBO of a resource is an additional output that can change from 0 to 1 but not back. This can be interpreted as a single event, which happens when the MBO changes to 1. We generalize this to many events by the means of a global event history.

**Definition 1.** *Let $\mathcal{N}$ be a name set. The* global event history *$\mathcal{E}$ is a list of elements of $\mathcal{N}$ without duplicates.*

*For $n \in \mathcal{N}$, we use $\mathcal{E}_n$ as a short-hand notation to denote that $n$ is in the list $\mathcal{E}$, and say that the event happened. Analogously, we use $\neg\mathcal{E}_n$ to denote the complementary case. Furthermore, we denote by $\mathcal{E} \xleftarrow{+} \mathcal{E}_n$, the act of appending $n$ to the list $\mathcal{E}$, if $\neg\mathcal{E}_n$, and leaving the list unchanged otherwise.*

We also introduce the natural happened-before relation on the events.

**Definition 2.** *For $n_1, n_2 \in \mathcal{N}$, we say that the event $n_1$ precedes the event $n_2$ in the event history $\mathcal{E}$, denoted $\mathcal{E}_{n_1} \prec \mathcal{E}_{n_2}$, if either*

- *both events happened, i.e, $\mathcal{E}_{n_1}$ and $\mathcal{E}_{n_2}$, and $n_1$ is in the history before $n_2$,*
- *or only $n_1$ happened so far.*

[1] From a conceptual point of view, this global event history is somewhat reminiscent of the "directory" ITI used in the recent version (as of December 2018) of UC [4] to keep track of which parties are corrupted.

Note that saying that $\mathcal{E}_{n_1} \prec \mathcal{E}_{n_2}$ is true if so far only the former one has happened best matches the type of statement we usually want to make: for instance, if we express the condition that a message is secure if the key has been securely erased before the memory was leaked, then we do not need to insist that the memory actually leaked.

**Event-Aware Systems.** We consider resources, converters and distinguishers that can (1) read the global event history, and (2) append to the event history from a fixed subset of $\mathcal{N}$. That is, the global event history is an additional component (of both the real and ideal world) that models event-awareness in an abstract manner, rather than formalizing them as outputs that need to be explicitly passed between components.

As a convention, we use as event-name pairs $(id, \mathsf{label})$, where $\mathsf{label}$ is a descriptive keyword (e.g., $\mathsf{leaked}$), and $id$ identifies the resource triggering the event, and we use the notation $\mathcal{E}_{id}^{\mathsf{label}}$. Simulators and distinguishers can trigger events with arbitrary $id$'s (looking forward, e.g. a simulator will have to trigger real-world events that do not occur in the ideal world). Still, we require that they do not trigger events that can be triggered by any resources they are connected to (such that, for example, a memory-leaked event really means that it did leak).

**Definition 3.** *A simulator is* compatible *if it only triggers events that cannot be triggered by the resource it is attached to. For two resources* $\mathsf{R}$ *and* $\mathsf{S}$*, a distinguisher* $\mathbf{D}$ *is* compatible *if it only triggers events that cannot be triggered by neither* $\mathsf{R}$ *nor* $\mathsf{S}$*.*

Converters implementing protocols, on the other hand, do not depend on the event history, since an event is something that *might* be observable, rather than something that is guaranteed to be observable by the honest parties.

**Construction Notion.** Intuitively, in the context of events, a real-world resource $\mathsf{R}$ is "just-as-good" as $\mathsf{S}$ if these resources look the same to distinguishers $\mathbf{D}^{\mathcal{E}}$ with read-and-write access to the global event history $\mathcal{E}$. This implies that the sequences of events must be the same in the real and in the ideal world. However, for convenience, we slightly relax this rule and introduce event renaming. For example, if a memory is used to store a key, then the memory-read event in the real world would have in the ideal world a better name key-received. Hence, we use both names to denote the same event (one can think of them as aliases). Moreover, we also allow for multiple aliases for a more fine-grained consideration of events in the ideal world, for instance by separating a message-received event into a successful and unsuccessful one.

We make this renaming explicit in the construction statements by defining a surjection $\tau$ that maps events triggered by the ideal-world resource to their real-world counterparts. (Note that in the case of duplicates caused by $\tau$, $\tau(\mathcal{E})$ only contains the first occurrence.) When referring to real-world events for specifying ideal-world guarantees, we will sometimes use $\tilde{\mathcal{E}} \coloneqq \tau(\mathcal{E})$ as a shorthand notation.

We can now define the construction notion for two resources with events.

**Definition 4.** *We say that* $(\mathsf{prot}_1, \mathsf{prot}_2)$ *constructs* $\mathsf{S}$ *from* $\mathsf{R}$ *under the event-renaming* $\tau$*, denoted*

$$\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R} \;\;\widehat{\approx}_{\tau}\;\; \mathsf{sim}^{\mathtt{E}}\,\mathsf{S},$$

*if there exists an efficient simulator* $\mathsf{sim}$*, such that* $\tau$ *only renames events triggered by* $\mathsf{sim}^{\mathtt{E}}\,\mathsf{S}$*, and for all efficient event-aware distinguishers* $\mathbf{D}^{\mathcal{E}}$*, compatible for* $\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}$ *and* $\mathsf{sim}^{\mathtt{E}}\,\mathsf{S}$ *the following advantage is negligible.*

$$\begin{aligned}\Delta^{\mathbf{D}^{\mathcal{E}}}\big(\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R},\ \mathsf{sim}^{\mathtt{E}}\,\mathsf{S}\big)&\\ := \Pr\big[\mathbf{D}^{\tau(\mathcal{E})}(\mathsf{sim}^{\mathtt{E}}\,\mathsf{S}) = 1\big] - \Pr\big[\mathbf{D}^{\mathcal{E}}(\mathsf{prot}_1^{\mathtt{A}}\,\mathsf{prot}_2^{\mathtt{B}}\,\mathsf{R}) = 1\big]&\end{aligned}$$

We stress that this construction notion satisfies the axioms of the more abstract layer on which the composition theorem of CC is proven [18,19], and thus composes as well.

## 4 Composable Guarantees for Secure Messaging

In this section we introduce the unified type of construction statement—in CC with events—that we make about SM protocols and components thereof.

### 4.1 The Approach

We opt for the natural choice of an *application-centric approach*, where the security of a cryptographic scheme or primitive is defined as the construction it achieves when used in a particular application. While this approach provides readily understandable and clean security statements, the resulting definitions often turn out to be overly specific. For instance, the statement about an encryption scheme might hard-code a particular assumed authentic communication network, implying that it cannot be directly combined with an authentication scheme achieving slightly different guarantees.

Avoiding such overly specific statements is crucial for a modular treatment of ratcheting protocols, as each sub-protocol of the prior literature achieves slightly different guarantees. We address this problem by making parameterized construction statements, where the assumed real-world resources are parameterized by several "switches" determining their security guarantees. Formally, such a "switch" is represented by a function of the global event history $\mathcal{E}$ (among others), that dynamically defines the behavior of the resource at a given moment in time. For instance, a leakage function $\mathfrak{L}$ may specify to which extent a channel leaks depending on the set of events that happened so far. The goal of a protocol is then expressed as improving certain parameters while leaving the others unchanged, independently of what they were in the beginning. That is, our construction statements will be of the type that a protocol constructs a communication network with certain (stronger) guarantees, assuming a network with certain (weaker) guarantees, where the real-world guarantees are treated as a parameter instead of hard-coding them.

Note that in the context of ratcheting protocols, making such parameterized statement about components—without a-priori assuming any guarantees about the real-world—is mostly not an issue. This is due to the fact that the protocols anyway have to be designed for the setting where the state and randomness could leak at any time, temporarily nullifying all guarantees that the component might try to assume from the underlying sub-protocols.

### 4.2 Our Channel Model

We now introduce our model of two-party communication networks. It allows us to express flexible security guarantees, but also various usability restrictions or guarantees, such as whether messages can be received out of order or not.

*Many Single-Message Channels.* We choose to model the communication network between Alice and Bob as the parallel composition of many unidirectional single-message communication channels. Besides being simpler to describe, it allows to have simpler construction steps which only consider a subset of the channels. On the flip side, it results in a world with an arbitrary but bounded number of messages, as the set of resources is static in CC. This is, however, without loss of generality as long as the protocols do not take advantage of this upper bound. Finally, observe that this decision results in a network where messages have implicit (unprotected) sequence numbers, as for instance achieved by TCP.

*The Single-Message Channel.* We model channels with authenticated data. Since we will use the same type of channel both in the real and ideal world, the channel must hit the right trade-off between giving enough power to the simulator but not too much power to the real-world adversary. On a high level, the channel interfaces and their capabilities are as follows. See Fig. 3 for the formal definition.

- The sender `S` can issue the command $(\texttt{send}, m, ad)$. Whether she is allowed to do so is determined by the can-send predicate $\mathfrak{S}$. (This predicate will mainly be used to describe situations in which the sender does not have the necessary keys yet.) A successful sending operation triggers the event $\mathcal{E}^{\mathsf{sent}}$. The sender can also query whether the channel is available for transmission.
- The adversary `E` can then potentially learn $m$ through the `read` command. Whether she is allowed to do so is determined by the can-leak function $\mathfrak{L}$, which outputs either `false` (the adversary is not allowed to read $m$), `true` (reading is allowed but triggers a leaked event $\mathcal{E}^{\mathsf{leaked}}$), or `silent` (reading is allowed). Moreover, she is always allowed to learn the length of $m$ and the (non-confidential) associated data $ad$.
- The adversary decides when receiving becomes possible, i.e., the message in principle is delivered. Once this happens, the receiver `R` can try to fetch the message. This has two possible outcomes: either he receives a message and an according received event is triggered, or he receives $\perp$ and an error event (indexed by an error code from Errors) is triggered. Which case happens is

**Resource** $\mathsf{Ch}^{\mathsf{id},\mathtt{S}\to\mathtt{R}}_{\mathfrak{L},\mathfrak{I},\mathfrak{S},\mathfrak{R},\mathsf{Errors}}$

**Parameters:**
- Identity $\mathsf{id}$ (optionally), and interfaces $\mathtt{S}$ (sender) and $\mathtt{R}$ (receiver)
- Set of $\mathsf{Errors}$ that can occur
- Functions $\mathfrak{L}(\mathcal{E}) \in \{\mathtt{true}, \mathtt{false}, \mathtt{silent}\}$ (can leak), $\mathfrak{S}(\mathcal{E}) \in \{\mathtt{true}, \mathtt{false}\}$ (can send), $\mathfrak{R}(\mathcal{E}) \in \{\mathtt{true}, \mathtt{false}\}$ (can receive) and $\mathfrak{D}(\mathcal{E}, same) \in \mathsf{Errors} \cup \{\mathtt{msg}\}$ (delivery outcome)

**Events:** $\mathcal{E}^{\mathsf{sent}}$, $\mathcal{E}^{\mathsf{leaked}}$, $\mathcal{E}^{\mathsf{received}(same)}$ and $\mathcal{E}^{\mathsf{error}(err,same)}$ for $same \in \{\mathtt{true}, \mathtt{false}\}$ and $err \in \mathsf{Errors}$

**Initialization**

$m_\mathtt{S}, ad_\mathtt{S}, m_\mathtt{R}, ad_\mathtt{R}, cmd, same \leftarrow \bot$

**Interface S**

**Input:** $(\mathtt{send}, m, ad) \in \mathcal{M} \times \mathcal{AD}$
**assume** $cmd = \bot$
**if** $\neg\mathfrak{S}(\mathcal{E})$ **then output** $\bot$
$(m_\mathtt{S}, ad_\mathtt{S}) \leftarrow (m, ad)$
$\mathcal{E} \xleftarrow{+} \mathcal{E}^{\mathsf{sent}}$
**output** $\mathtt{ok}$

**Input:** $\mathtt{isAvailable}$
**output** $\mathfrak{S}(\mathcal{E})$

**Interface R**

**Input:** $\mathtt{receive}$
**assume** only called once
**if** $\neg\mathfrak{R}(\mathcal{E}) \vee cmd = \bot$ **then**
  **output** $\bot$

// same messages (no injection)?
**if** $same = \mathtt{check}$ **then**
  $same \leftarrow ((m_\mathtt{R}, ad_\mathtt{R}) = (m_\mathtt{S}, ad_\mathtt{S}))$

// the outcome: an error or the message
$out \leftarrow \mathfrak{D}(\mathcal{E}, same)$
**if** $cmd = \mathtt{dlv} \wedge out = \mathtt{msg}$ **then**
  $\mathcal{E} \xleftarrow{+} \mathcal{E}^{\mathsf{received}(same)}$
  **output** $(m_\mathtt{R}, ad_\mathtt{R})$
**else if** $cmd = (\mathtt{err}, err, Overw)$
    $\wedge (out = \mathtt{msg} \vee out \in Overw)$ **then**
  $\mathcal{E} \xleftarrow{+} \mathcal{E}^{\mathsf{error}(err,same)}$
  **output** $\bot$
**else**
  $\mathcal{E} \xleftarrow{+} \mathcal{E}^{\mathsf{error}(out,same)}$
  **output** $\bot$

**Input:** $\mathtt{isAvailable}$
**output** $\mathfrak{R}(\mathcal{E}) \wedge cmd \neq \bot$

**Interface E**

**Input:** $\mathtt{read}$
**if** $\mathfrak{L}(\mathcal{E}) = \mathtt{false}$ **then output** $\bot$
**else if** $\mathfrak{L}(\mathcal{E}) = \mathtt{true}$ **then** $\mathcal{E} \xleftarrow{+} \mathcal{E}^{\mathsf{leaked}}$
**output** $(m_\mathtt{S}, ad_\mathtt{S})$

**Input:** $\mathtt{readLength}$
**output** $(|m_\mathtt{S}|, ad_\mathtt{S})$

**Input:** $(\mathtt{deliver}, m, ad, same') \in (\mathcal{M} \cup \{\mathtt{fwd}\}) \times \mathcal{AD} \times \{\mathtt{check}, \mathtt{false}\}$
**assume** $cmd = \bot$

// handle forwarding request
**if** $m = \mathtt{fwd}$ **then**
  **if** $m_\mathtt{S} = \bot$ **then output** $\bot$
  **else** $m \leftarrow m_\mathtt{S}$

// store for receiving
$(m_\mathtt{R}, ad_\mathtt{R}, same) \leftarrow (m, ad, same')$
$cmd \leftarrow \mathtt{dlv}$
**output** $\mathtt{ok}$

**Input:** $(\mathtt{error}, err, Overw, m, ad, same') \in \mathsf{Errors} \times 2^{\mathsf{Errors}} \times \mathcal{M} \times \mathcal{AD} \times \{\mathtt{true}, \mathtt{false}, \mathtt{check}\}$
**assume** $cmd = \bot$

// $(m, ad)$ only to determine same
**if** $same' = \mathtt{check}$ **then**
  $(m_\mathtt{R}, ad_\mathtt{R}) \leftarrow (m, ad)$

// store for receiving
$same \leftarrow same'$
$cmd \leftarrow (\mathtt{err}, err, Overw)$
**output** $\mathtt{ok}$

**Fig. 3.** The single-message channel.

determined by the delivery function $\mathfrak{D}$, which takes into account the event history and on whether the message that $\mathtt{R}$ tries to fetch is the same as the one input by $\mathtt{S}$ (or an injected value from the adversary). The latter condition is denoted by the flag *same*. The flag *same* is also exposed as part of the received or error event $\mathcal{E}^{\mathsf{received}(same)}$ or $\mathcal{E}^{\mathsf{error}(err,same)}$, respectively.

– When the adversary decides that receiving is possible, she has two options: schedule the delivery of $(m', ad')$ (command `deliver`), or force an error $err \in \mathsf{Errors}$ to be triggered (command `error`). In the first case, she can also request to just forward the sender's message (if one exists), using $m' = \texttt{fwd}$. Moreover, for technical reasons[2], she can also insist that once the receiver fetches the message, $same = \texttt{false}$ is used even if the messages match. In case the adversary forces an error *err* and the outcome of receiving would anyway be a (different) error, the existing error can either be overwritten or preserved. She can control this by specifying a set *Overw* of errors that should be overwritten.

*A Note on Confidentiality.* In our channel, the $\mathcal{E}^{\mathsf{received}(same)}$ and $\mathcal{E}^{\mathsf{error}(err,same)}$ events indicate whether the message that Eve injected was the same as the sender's. Since we assume that those events are in principal observable by everybody, including the adversary, those events can partially breach confidentiality if the communication is not properly authenticated.

However, those events are crucial to phrase the post-impersonation guarantees of certain ratcheting protocols. In fact, in those protocols Eve could usually inject her own message (after exposing the sender's state), observe whether it causes the communication to break down, and thereby deducing whether the sender wanted to send the same message afterwards. Our events simply reflect this.

### 4.3 Additional Resources: Memory and Randomness

An integral part of secure messaging protocols is the assumption that the parties' state, and sometimes also randomness, can leak to the adversary. In Constructive Cryptography everything that can be accessible by multiple parties, here the honest party and Eve, must be modeled as a resource. As a consequence, all of our converters will be stateless and deterministic. (Stateless means that the converter cannot keep state between two separate invocations at the emulated interfaces.) The statements will contain explicit memory and randomness resources instead.

We consider two types of memory resources: (1) an insecure memory $\mathsf{IMem}^{id,\mathtt{U}}$, and (2) a potentially secure memory $\mathsf{Mem}^{id,\mathtt{U}}$. The main differences are that the latter one can be securely erased at any time, is parameterized in a can-leak predicate $\mathfrak{L}$, and triggers a leaked event $\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(id,U)}$ once the content actually leaks to the adversary. Since each event can only occur once, we thus model it as a write-once memory. Rewritable secure memory can then be modeled as the parallel composition of many write-once memory cells,[3] where each can be

[2] The simulator might need this capability, e.g., if two (abstracted away) ciphertexts decrypt to the same message. Note that providing additional capabilities to the adversary in the real world only strengthens the statement and directly implies the construction where this capability is removed.

[3] The memory requirement of a protocol is not determined by the number of such write-once memories, but rather by the maximal number of them in use at any time.

leaked independently[4]. Analogously, the randomness resource is parameterized by a predicate $\mathfrak{L}$ as well. If allowed, the randomness can leak (triggering $\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(id)}$) to the adversary at the moment it is used by the honest party—modeling that it is sampled fresh at this point and is not stored. See the full version [11] for a formal definition of both resources.

## 5 Unifying Ratcheting: Two Examples

In this section, we get acquainted with how the security guarantees of ratcheting protocols can be phrased within our model. To this end, we model the guarantees of two components of actual ratcheting protocols.

As a first example, we consider a simple authentication scheme that appears in [8–10]. Using this example, we demonstrate how our framework allows for a fine-grained modularization, with the overall security then directly following from composition. As a second example, we consider the use of hierarchical identity-based encryption, as in [9,24]. In this example, we explore a way to work around the so-called commitment issue of composable security.

### 5.1 A Simple Authentication Scheme

We first consider a simple unidirectional authentication protocol, which is designed with the strong guarantees of secure messaging in mind: the authentication guarantees should not only be forward secure but also heal after a state or randomness exposure of either party. Slight variations of this protocol have been used in [10] (without the hash) and [8] (using signcryption). Essentially the same idea also appeared in [9], where, however, a stronger signature primitive with updatable keys is considered, leading to the protocol being formalized in the bidirectional setting.

**The Protocol.** In the protocol, whenever the sender wants to send a message, a fresh signing and verification key pair is sampled. The fresh verification key is then signed together with the message—using the prior signing key—and the message, the verification key and the signature are transmitted. Finally, the old signing key is securely erased and the fresh one stored instead. The receiver, on the other hand verifies a received message with the previous verification key and stores the new one. The scheme is depicted in Fig. 4.

Recall that we aim to make a strong construction statement that considers how the scheme enhances any preexisting security guarantees, including confidentiality. Usually preserving confidentiality is not a goal that is considered for an authentication protocol, moreover, it is known that the authenticate-then-encrypt approach used in old versions of TLS is not generally secure [13]. Nevertheless, we show that the scheme actually achieves this at the cost of assuming

[4] Technically this leads to more fine-grained statements compared to prior work where it was usually assumed that either the entire state leaks or not. Nevertheless, it does not appear to incur additional significant complications.

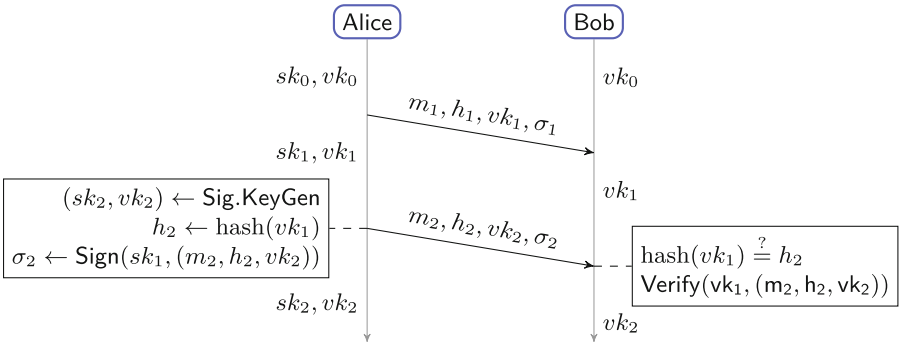


**Fig. 4.** The simple scheme for unidirectional authentication.

unique signatures instead of unforgeable ones (analogous to [9]), and with a minor modification: with each message, the sender also transmits a hash of the previous verification key. Such a hash is also present in the protocol from [9], and allows the receiver to check whether he is using the correct verification key.

**The Guarantees.** Clearly, the protocol achieves authenticity if neither party's state is exposed. Moreover, Bob's state only consists of public information. If Alice's state gets exposed, then Eve obtains her current signing key that she can use to impersonate Alice towards Bob at this point in time. However, this key is useless to tamper with previous messages, even if they have not been delivered yet (forward security). More importantly, if, for some reason, Alice's next message containing a fresh verification key still is delivered without modification, then the signing key obtained by the adversary becomes useless thereby achieving the healing property. Hence, the adversary can inject the $i$-th message if and only if Alice's state between the $(i-1)$-st and $i$-th message got exposed, or there has already been a successful injection before.

Expressing the scheme's security guarantees in a game-based manner turned out to be surprisingly involved compared to the scheme's simplicity and how easy it seems to intuitively describe its guarantees. Notably, to show its security, in [10] the abstraction of a key-updating signature scheme, as well as its corresponding correctness and security games, have been introduced. This raises a couple of questions: can't we do simpler? What is the right security statement to make about this quite simple protocol, and what happens if the channel already provides certain authenticity or confidentiality guarantees? In the following, we try to answer these questions.

**The Construction Statement.** First, note that we consider the authentication of messages directly, and do not introduce an intermediate signature notion.

Secondly, we consider authenticating the $i$-th message only, and to this end consider the $(i-1)$-st message where the fresh verification key is transmitted (we do not authenticate this message here) and the $i$-th message that is then signed under the corresponding signing key. Authenticating the $(i-1)$-st message, and all others, is then taken care of by iteratively applying the protocol, with the overall security directly implied by the composition theorem. This leads to the following real world resources

$$\mathsf{R}_i^{\mathsf{auth}} := \left[\mathsf{Ch}^{i-1,\mathtt{A}\to\mathtt{B}}, \mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}, \mathsf{Rand}^{kg_i,\mathtt{A}}, \mathsf{Mem}^{sk_i,\mathtt{A}}, \mathsf{IMem}^{vk_i,\mathtt{B}}\right], \tag{1}$$

where besides the two channels the sender also has a memory to store the new signing key, and the receiver a (insecure) memory to store the verification key. Furthermore, the sender also has an explicit randomness resource available (note that we only need key-generation randomness, since unique signatures are deterministic). The corresponding protocol converters $(\mathsf{sig}_i, \mathsf{vrf}_i)$ that are connected to Alice's and Bob's interfaces of $\mathsf{R}_i^{\mathsf{auth}}$, respectively, simply implement the previously described protocol. A formal description of those protocol converters can be found in the full version [11].

The goal of the protocol is then phrased as constructing the following ideal-world resource

$$\mathsf{S}_i^{\mathsf{auth}} := \left[\mathsf{Ch}^{i-1,\mathtt{A}\to\mathtt{B}}, \mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}\right], \tag{2}$$

in which the channels can also trigger an error $\mathsf{sig\text{-}err}$, indicating that the signature verification failed, in addition to the errors from the real-world counterparts.

The authentication guarantees for the $i$-th channel can then be expressed via the following delivery-function, which guarantees that an injection attempt ($\neg same$) when the key is not known will causes a signature-verification error $\mathsf{sig\text{-}err}$, and preserves preexisting authenticity (recall that $\tilde{\mathcal{E}} := \tau(\mathcal{E})$ denotes the real-world's event history):

$$\mathfrak{D}^{\mathsf{S}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\mathcal{E}, same) := \begin{cases} err & \text{if } \mathfrak{D}^{\mathsf{R}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\tilde{\mathcal{E}}, same) = err \wedge err \neq \mathtt{msg} \\ \mathtt{msg} & \text{else if } same \vee \mathcal{E}_i^{\mathsf{sk\text{-}known}} \\ \mathsf{sig\text{-}err} & \text{else} \end{cases} \tag{3}$$

where in a slight abuse of notation, we define a composed event $\mathcal{E}_i^{\mathsf{sk\text{-}known}}$, which is triggered as soon as it is not excluded that the signing key corresponding to Bob's verification key is known to Eve:

$$\mathcal{E}_i^{\mathsf{sk\text{-}known}} := \mathcal{E}^{\mathsf{injected}}_{\mathsf{Ch}(i-1,\mathtt{A}\to\mathtt{B})} \vee \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(kg_i,\mathtt{A})} \vee \left(\mathcal{E}^{\mathsf{sent}}_{\mathsf{Ch}(i-1,\mathtt{A}\to\mathtt{B})} \prec \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(sk_i,\mathtt{A})} \prec \mathcal{E}^{\mathsf{sent}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}\right).$$

On the flip side, the scheme limits the availability of the channels to be sequential. While sending messages in order is natural for Alice, the protocol restricts Bob to receive them in order as well. We can express this using the following predicates.

$$\mathfrak{S}^{\mathsf{S}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\mathcal{E}) := \mathfrak{S}^{\mathsf{R}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\tilde{\mathcal{E}}) \ \wedge \ \mathcal{E}^{\mathsf{sent}}_{\mathsf{Ch}(i-1,\mathtt{A}\to\mathtt{B})}, \tag{4}$$

$$\mathfrak{R}^{\mathsf{S}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\mathcal{E}) := \mathfrak{R}^{\mathsf{R}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\tilde{\mathcal{E}}) \ \wedge \ \mathcal{E}^{\mathsf{received}}_{\mathsf{Ch}(i-1,\mathtt{A}\to\mathtt{B})}. \tag{5}$$

Note that our model simply forces us to make this restriction explicit, whereas this is often just hard-coded in games.[5]

All other parameters and predicates are preserved, e.g. $\mathfrak{L}^{\mathsf{S}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\mathcal{E}) := \mathfrak{L}^{\mathsf{R}_i^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\tilde{\mathcal{E}})$. The security of the protocol can then be phrased as constructing the ideal world $\mathsf{S}_i^{\mathsf{auth}}$ from the real world $\mathsf{R}_i^{\mathsf{auth}}$, as summarized in the following theorem.

**Theorem 1.** *Let* $\mathsf{R}_i^{\mathsf{auth}}$ *be as in* (1)*, and let* $\mathsf{S}_i^{\mathsf{auth}}$ *be as in* (2)*, with the guarantees and restrictions as described in* (3)*,* (4)*, and* (5)*, respectively, and all others guarantees unchanged from* $\mathsf{R}_i^{\mathsf{auth}}$*. Moreover, let* $\tau$ *map the event* $\mathcal{E}^{\mathsf{error}(\mathsf{sig\text{-}err},same)}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$ *to* $\mathcal{E}^{\mathsf{received}(same)}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$*. Then there exists an efficient simulator* $\mathsf{sim}$ *such that*

$$\mathsf{sig}_i^{\mathtt{A}} \ \mathsf{vrf}_i^{\mathtt{B}} \ \mathsf{R}_i^{\mathsf{auth}} \ \approx_\tau \ \mathsf{sim}^{\mathtt{E}} \ \mathsf{S}_i^{\mathsf{auth}},$$

*if the underlying signature scheme is unforgeable with unique signatures, and the hash function is collision resistant.*

*Proof.* The proof is found in the full version [11]. Note that compared to a normal signature-scheme proof it is quite involved, which is the main price we pay for our much stronger statement.

**Extending to Many Messages.** So far, we only considered a world where Alice sends two messages, of which the second is authenticated. In a realistic setting, Alice can of course send many messages where all of them should be authenticated. In this section, we see how the composition theorem of Constructive Cryptography can be applied to directly get the desired result.

In particular, we start with a sequence of possibly unauthenticated channels $\mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}$ for $i \in [n]$, where the authentication of $\mathsf{Ch}^{0,\mathtt{A}\to\mathtt{B}}$ can be seen as a setup assumption (it is standard to assume that Alice and Bob initially share a signing-verification key pair). Then, we iteratively apply the construction for two channels to $\mathsf{Ch}^{0,\mathtt{A}\to\mathtt{B}}$ and $\mathsf{Ch}^{1,\mathtt{A}\to\mathtt{B}}$, then to $\mathsf{Ch}^{1,\mathtt{A}\to\mathtt{B}}$ and $\mathsf{Ch}^{2,\mathtt{A}\to\mathtt{B}}$, etc. (c.f. Fig. 5). The composition theorem of CC guarantees that the composed protocol constructs the ideal world.

**Corollary 1.** *Let* $\mathsf{R}^{\mathsf{auth}}$ *and* $\mathsf{S}^{\mathsf{auth}}$ *denote the following real and ideal worlds*

$$\mathsf{R}^{\mathsf{auth}} := \left[\left\{\mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}\right\}_{i\in\{0,\ldots,n\}}, \left\{\mathsf{Mem}^{sk_i,\mathtt{A}}, \mathsf{IMem}^{vk_i,\mathtt{B}}\right\}_{i\in[n]}\right],$$

[5] Actually, many recently proposed secure-messaging protocols do have this restriction, which might limit their usability as pointed out by [1].

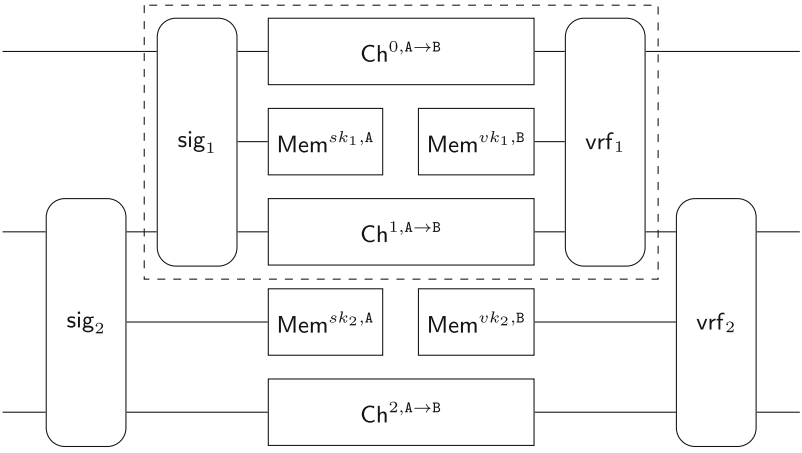


**Fig. 5.** The first two steps constructing a sequence of authenticated channels: (1) The protocol $(\mathsf{sig}_1, \mathsf{vrf}_1)$ constructs a hybrid world, where the resources in the dashed box are replaced by two channels $\mathsf{Ch}^{0,\mathtt{A}\to\mathtt{B}}$ and $\mathsf{Ch}^{1,\mathtt{A}\to\mathtt{B}}$, where $\mathsf{Ch}^{1,\mathtt{A}\to\mathtt{B}}$ is authenticated as long as $\mathsf{Ch}^{0,\mathtt{A}\to\mathtt{B}}$ is. (2) $(\mathsf{sig}_2, \mathsf{vrf}_2)$ constructs the ideal world, where $\mathsf{Ch}^{1,\mathtt{A}\to\mathtt{B}}$ and $\mathsf{Ch}^{2,\mathtt{A}\to\mathtt{B}}$ are authenticated as long as $\mathsf{Ch}^{0,\mathtt{A}\to\mathtt{B}}$ is.

*and*

$$\mathsf{S}^{\mathsf{auth}} := \left[\left\{\mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}\right\}_{i\in\{0,\ldots,n\}}\right],$$

*respectively. Then, there exists an efficient simulator* $\mathsf{sim}$ *such that*

$$(\mathsf{sig}_1,\ldots,\mathsf{sig}_n)^{\mathtt{A}}\ (\mathsf{vrf}_1,\ldots,\mathsf{vrf}_n)^{\mathtt{B}}\ \mathsf{R}^{\mathsf{auth}} \ \approx\ \mathsf{sim}^{\mathtt{E}}\mathsf{S}^{\mathsf{auth}},$$

*where for each* $i \in [n]$, $\mathfrak{I}^{\mathsf{S}^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$, $\mathfrak{S}^{\mathsf{S}^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$, *and* $\mathfrak{R}^{\mathsf{S}^{\mathsf{auth}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$ *are defined as in* (3), (4), *and* (5), *respectively.*

### 5.2 Confidentiality from HIBE

In the following we discuss a protocol from [9] that uses *hierarchical identity-based encryption (HIBE)* to add confidentiality to a sequence of channels. The protocol was designed for a challenging setting, where we do not assume authentication (as is usually done when talking about encryption). The reason is that in secure messaging authentication cannot be guaranteed when the sender's state is exposed. This situation fits perfectly to our framework.

The protocol is described in the so-called sesqui-directional setting, introduced in [24], meaning that the messages from both directions are considered, but only the guarantees of one of the directions are under concern—here from Alice to Bob. The bidirectional guarantees then follow directly from composition.

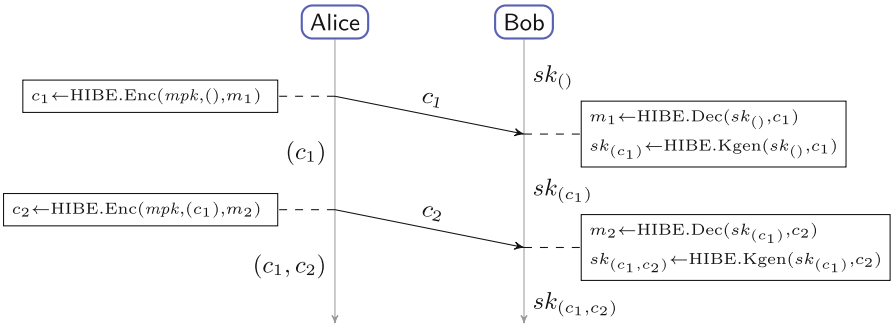


**Fig. 6.** The first epoch of the sesquidirectional HIBE protocol.

**Hierarchical Identity-Based Encryption.** A HIBE scheme consists of the following four algorithms:

- A setup generation algorithm $(mpk, msk) \leftarrow \text{HIBE.Setup}(1^\kappa; r)$, generating the root master public and secret keys, i.e. $sk_{()} = msk$.
- A key-generation algorithm $sk_{\text{id}\|id_n} \leftarrow \text{HIBE.Kgen}(sk_{\mathbf{id}}, id_n)$, where $(\mathbf{id} \parallel id_n) \coloneqq (id_1, \ldots, id_{n-1}, id_n)$ for an identity vector $\mathbf{id} = (id_1, \ldots, id_{n-1})$.
- An encryption algorithm $c \leftarrow \text{HIBE.Enc}(mpk, \mathbf{id}, m; r)$.
- A decryption algorithm $m \leftarrow \text{HIBE.Dec}(sk_{\mathbf{id}}, c)$.

We require the HIBE scheme to be IND-CCA secure with certain additional properties that are not guaranteed by IND-CCA itself, but that most schemes do provide (see the full version [11] for details).

**The Protocol Overview.** On a high level, the protocol proceeds in epochs, where in each epoch Bob sends one message to Alice, and then Alice sends a sequence of messages to Bob. In particular, Bob's message contains a fresh HIBE public key $mpk$. For simplicity, consider the first epoch, as depicted in Fig. 6. When Alice sends her $i$-th message, she encrypts it with $mpk$, using as the identity (the hashes of) all ciphertexts she sent before. Whenever Bob receives a ciphertext $c_i$, he decrypts it, derives the secret key for the new identity (with $c_i$ appended) and erases the old key.

In the next epoch, Bob sends a new public key $mpk'$, and we repeat. One subtle issue is how to run the epochs together. Note that, for example, Bob may send a number of public keys without receiving a response, in which case he has to store secret keys from a number of epochs. A fresh secret key is stored for the empty identity, and when Bob receives a ciphertext, he updates all currently stored secret keys. This means that Alice uses for encryption of the $i$-th message a truncated transcript $(c_r, \ldots, c_{i-1})$. In order for her to compute it, Bob sends with each public key the index $r$ of the last message he received.

**Security Intuition.** Intuitively, this use of HIBE allows to achieve three goals. The first is healing, achieved by exchanging fresh keys, as in most secure-messaging schemes. The second is forward secrecy: exposing the secret key after the $i$-th message is received does not affect the confidentiality of messages $m_1, \ldots, m_{i-1}$. This holds, since Bob updated all the secret keys with the identity $c_i$ in the meantime. Healing and forward secrecy could also be achieved by a forward-secure PKE scheme. The last goal is the so-called post-impersonation security: an active injection destroys the decryption keys, so that its leakage exposes no messages. For this we need the hierarchy of identities. Roughly, injecting a message $c_i'$ causes Bob to update his key to $sk_{(c_r,\ldots,c_i')}$. This key gives no information about messages encrypted by Alice, since those will be for another identity $(c_r, \ldots, c_i)$.

**The Construction Statement.** To formalize these guarantees as a construction statement, we first have to describe the real world in which the protocol is executed. It consists of $n$ channels from Alice to Bob (which the protocol protects) and $n$ channels in the opposite direction on which the master public keys are transmitted. Moreover, Alice has memories to store the public keys and the transcript, and randomness resources for the encryption. Bob, on the other hand, has memories to store the secret keys and randomness resources for the key generation:

$$\mathsf{R}^{\mathsf{hibe}} := \Big[\big\{\mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}\big\}_{i\in[n]}, \big\{\mathsf{Ch}^{j,\mathtt{B}\to\mathtt{A}}\big\}_{j\in[n]}, \mathsf{IMem}^{pk,\mathtt{A}}, \big\{\mathsf{Rand}^{kg_j,\mathtt{B}}\big\}_{j\in[n]}, \big\{\mathsf{Mem}^{tr_i,\mathtt{A}}, \mathsf{Rand}^{enc_i,\mathtt{A}}\big\}_{i\in[n]}, \big\{\mathsf{Mem}^{sk_{(j,i)},\mathtt{B}}\big\}_{j\in[n],i\in[n+1]\}}\Big], \quad (6)$$

where the index $i$ indicates that the resource is related to transmitting the $i$-th message from Alice to Bob, and the index $j$ indicates the $j$-th epoch. A formal description of the pair of converters implementing the protocol $(\mathsf{hibe\text{-}enc}, \mathsf{hibe\text{-}dec})$ can be found in the full version [11].

The goal of the protocol is to enhance the confidentiality of the channels. Thus, the same set of channels is present in the ideal world, while the memory and randomness resources are used up:

$$\mathsf{S}^{\mathsf{hibe}} := \Big[\big\{\mathsf{Ch}^{i,\mathtt{A}\to\mathtt{B}}\big\}_{i\in[n]}, \big\{\mathsf{Ch}^{j,\mathtt{B}\to\mathtt{A}}\big\}_{j\in[n]}\Big]. \quad (7)$$

Moreover, the ideal channels can trigger an additional error $\mathsf{dec\text{-}err}$, indicating that decryption failed (this error event corresponds to the real-world delivery event when the adversary injects an invalid ciphertext).

We now proceed to formalize the confidentiality guarantees of $\mathsf{S}^{\mathsf{hibe}}$ by defining in which situations the $i$-th message might be known to the adversary:

*The randomness leaked:* If the encryption randomness leaked to the adversary, i.e., $\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(enc_i,\mathtt{A})}$, then no PKE scheme can provide (full) confidentiality.

*The master public key was set by Eve:*] If Alice encrypts using a master public key (potentially) set by Eve, Eve can trivially decrypt. That is, if Alice used the $j$-th master public key and $\mathcal{E}^{\mathsf{injected}}_{\mathsf{Ch}(j,\mathtt{B}\to\mathtt{A})}$.

*The secret key leaked:* Assume Alice sent the $i$-th message during the $j$-th epoch, and let $sk_{(j,i)}$ denote the secret key that Bob uses to decrypt that message. If Eve learned $sk_{(j,i)}$, the message is obviously not confidential, which either happens if the randomness used to generate the master secret key leaked or a key that allows to compute $sk_{(j,i)}$ leaked from Bob's memory:

$$\mathcal{E}^{\mathsf{sk\text{-}leaked}}_{i,j} \coloneqq \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(kg_j,\mathtt{B})} \vee \exists k \in [r_j, i] : \left(\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(sk_{(j,k)},\mathtt{B})} \wedge \forall \ell \in [r_j, k] : \neg\mathcal{E}^{\mathsf{injected}}_{\mathsf{Ch}(\ell,\mathtt{A}\to\mathtt{B})}\right),$$

where $r_j$ denotes the first message Bob received after sending the $j$-th public key ($r_j$ is determined by the sent and received events in $\mathcal{E}$). Note that the last condition explicitly encodes the *post-impersonation guarantee*, meaning that $sk_{(j,k)}$ is only useful as long as Eve did not destroy it by injecting her own ciphertext. Forward-secrecy and healing, on the other hand, are encoded implicitly by the order in which those events can happen in the real world. We can make them more explicit by observing

$$\mathcal{E}^{\mathsf{sk\text{-}leaked}}_{i,j} \iff \mathcal{E}^{\mathsf{sent}}_{\mathsf{Ch}(j,\mathtt{B}\to\mathtt{A})} \prec \mathcal{E}^{\mathsf{sk\text{-}leaked}}_{i,j} \prec \mathcal{E}^{\mathsf{received}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})},$$

where the former condition denotes healing and the latter forward-secrecy.

In summary, we can define the following event denoting that the $i$-th message is insecure

$$\mathcal{E}^{\mathsf{exposed}}_{i} \coloneqq \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(enc_i,\mathtt{A})} \vee \mathcal{E}^{\mathsf{injected}}_{\mathsf{Ch}(j_i,\mathtt{B}\to\mathtt{A})} \vee \mathcal{E}^{\mathsf{sk\text{-}leaked}}_{i,j_i},$$

where $j_i$ denotes the epoch in which the $i$-th message has been sent (which is computable from the order of events in $\mathcal{E}$), leading to

$$\mathfrak{L}^{\mathsf{S}^{\mathsf{hibe}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}(\mathcal{E}) \coloneqq \begin{cases} \mathtt{silent} & \text{if } \mathcal{E}^{\mathsf{exposed}}_{i} \\ \mathtt{false} & \text{otherwise.} \end{cases} \tag{8}$$

Notice that the above can-leak function fully overwrites any real-world guarantees, and silences the leaked events. This is because in the protocol Alice stores the communication transcript. As a consequence, when her memory leaks, the ciphertext leaks as well, even if the assumed channel was in fact confidential. Moreover, this leakage does not correspond to the channel leaked event.

Analogous to the authentication scheme of the previous section, the HIBE scheme also limits the availability of the channels to be sequential, due to the hash-transcript used as identities. Moreover, Alice can obviously only encrypt using master public keys she received the public key. This could be made formal using the can-send and can-receive predicates $\mathfrak{S}$ and $\mathfrak{R}$, respectively.

**Working Around the Commitment Problem.** As described so far, the real and ideal world $\mathsf{hibe\text{-}enc}^{\mathsf{A}}\mathsf{hibe\text{-}dec}^{\mathsf{B}}\mathsf{R}^{\mathsf{hibe}}$ and $\mathsf{sim}^{\mathsf{E}}\mathsf{S}^{\mathsf{hibe}}$, respectively, are easily distinguishable for any simulator $\mathsf{sim}$. The issue is the so-called *commitment problem* of simulation based cryptography: if the distinguisher chooses to first see a ciphertext and then leak the corresponding decryption key, this cannot be simulated, since the simulator first has to output a fake ciphertext, before getting to know the message, and then explain it by outputting a corresponding decryption key. For normal PKE, and especially HIBE, schemes this is impossible.

One solution would be to consider static memory corruptions, where the set of states that can be leaked to the adversary is a parameter of the construction statement. Such a static guarantee is however weaker than the existing game-based definitions and, thus, thwarts our goal of developing a unified model to express the guarantees obtained by existing protocols. We thus opt for the alternative solution to strengthen the real world analogous to how the games disable certain oracles to prevent trivial impossibilities. To this end, we disallow the adversary from obtaining the secret key $sk_{(j,i)}$ if this would allow to trivially identify a fake ciphertext. That is, we assume

$$\mathfrak{L}^{\mathsf{R}^{\mathsf{hibe}}}_{\mathsf{Mem}(sk_{(j,i)},\mathtt{B})}(\mathcal{E}) \coloneqq \neg\exists k > i : \left(\mathcal{E}^{\mathsf{committed}}_{k,j} \prec \mathcal{E}^{\mathsf{exposed}}_{k}\right), \tag{9}$$

where $\mathcal{E}^{\mathsf{committed}}_{i,j}$ denotes the event that the simulator commits on the $i$-th ciphertext, and that it was encrypted under $mpk_j$. More concretely, this happens if the distinguisher

- explicitly asked for the ciphertext;
- requested a hash-transcript that depends on the ciphertext;
- requested a secret key for which the identity depends on the ciphertext;
- actively injected a ciphertext that got decrypted under a secret key whose identity depends on the ciphertext under consideration,

leading to the following definition

$$\mathcal{E}^{\mathsf{committed}}_{i,j} \coloneqq (j_i = j) \wedge \Big(\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})} \vee \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(tr_i,\mathtt{A})} \vee \Big(\neg\mathcal{E}^{\mathsf{injected}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})} \wedge \exists k \geq i : \left(\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(sk_{(j,k)})} \vee \mathcal{E}^{\mathsf{injected}}_{\mathsf{Ch}(k,\mathtt{A}\to\mathtt{B})}\right)\Big)\Big),$$

where again $j_i$ denotes the epoch in which the $i$-th message has been sent.

While the construction statement loses its evident executional semantics making those restrictions of the real world—it is no longer apparent what guarantees one gets when executing the protocol in the actual world where the memory leakage is obviously not restricted like this—it is analogous to game-based notions where the adversary has to choose beforehand whether a message is a challenge (and then prevents leaking the corresponding randomness or secret keys), or is an insecure message just to advance the state. Phrasing it in a composable framework, however, still has the advantage of modularity and reusability, that is, each subprotocol can be proven secure independently and the overall security directly following from the composition theorem.

**Summary and Analysis.** The HIBE-based scheme achieves the so far described construction, with one exception: to provide more power to the simulator and make the construction statement provable, we need to silence the real-world channels' leaked events after the message is exposed, i.e, $\mathfrak{L}^{\mathsf{R}^{\mathsf{hibe}}}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$ is arbitrary, except that if $\mathcal{E}_i^{\mathsf{exposed}}$, it no longer evaluates to `true`.[6]

Observe that while having to silence the leakage event in the real world limits reusability, the statement for instance is still generic enough to be composed with the authentication scheme from the previous section: if the real world is restricted like this (in the end, those events are just a mean to phrase dependencies and carry no real semantics), then the signature scheme, which preserves the can-leak predicate, and afterwards the HIBE scheme can be applied.

Overall, we have the following theorem, proven in the full version [11].

**Theorem 2.** *Let* $\mathsf{R}^{\mathsf{hibe}}$ *be as in* (6) *with the restrictions to work around the commitment-problem from* (9) *and the restriction described above, and let* $\mathsf{S}^{\mathsf{hibe}}$ *be as in* (7) *with the confidentiality guarantees from* (8)*, and in-order sending and receiving. Let* $\tau$ *map the event* $\mathcal{E}^{\mathsf{error}(\mathsf{dec\text{-}err},same)}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$ *to* $\mathcal{E}^{\mathsf{received}(same)}_{\mathsf{Ch}(i,\mathtt{A}\to\mathtt{B})}$*. Then there exists an efficient simulator* $\mathsf{sim}$*, such that*

$$\mathsf{hibe\text{-}enc}^{\mathtt{A}}\,\mathsf{hibe\text{-}dec}^{\mathtt{B}}\,\mathsf{R}^{\mathsf{hibe}} \;\;\hat{\approx}_{\tau}\;\; \mathsf{sim}^{\mathtt{E}}\,\mathsf{S}^{\mathsf{hibe}},$$

*if the HIBE scheme is IND-CCA secure with our additional assumptions.*

## 6 Adaptive Security

All protocols considered so far, and most of the ones in the literature, only achieve a weakened construction statement, where, due to the commitment problem, we assume that certain sequences of events cannot occur in the real world. Intuitively, this means that the adversary is somewhat static: for example, when she decides to see the contents of a channel (the ciphertext, in the real world), at the same time she decides that she will not look at the contents of certain memory (the secret key). While this is exactly what the standard game-based definitions guarantee, when expressed in a composable framework, it seems rather unsatisfactory.

Hence, in this section, we consider SM schemes that tolerate a fully adaptive adversary, i.e, allow to "explain" ciphertexts whenever needed due to leakage of secret keys. In particular, we present a technique that, given an SM protocol that suffers from the commitment problem, allows to construct an adaptive SM (ASM) protocol with almost the same guarantees, but that achieves fully adaptive security. This comes at the cost of efficiency and being able to send only a fixed number of messages without interaction. Applied to protocols with optimal security [9,24], our technique enables even stronger guarantees.[7] As an example, we apply it to the HIBE protocol from Sect. 5.2.

[6] This doesn't affect $\mathcal{E}^{\mathsf{committed}}_{i,j}$, that only considers leakage events before $\mathcal{E}_i^{\mathsf{exposed}}$.

[7] In game-based definitions, one can think of the "corrupt" oracle not being silenced even if the challenge has been issued, but instead outputting the secret state corresponding to the challenge bit 0.

Note that while the technique we use is essentially a general compiler that "removes" the commitment problem, formally phrasing such a theorem would be rather cumbersome for at least two reasons. First, there is not just one game-based definition of an SM scheme that could be lifted and, second, we require the specific simulation technique encoded in most game-based definitions, in contrast to the existential simulator of our constructive SM statements.

### 6.1 Overview

*Receiver Non-committing Encryption.* The technical tool we use to construct adaptively-secure secure-messaging (ASM) schemes with optimal security is so-called receiver non-committing encryption (RNCE), introduced by Canetti et al. [5]. Intuitively, in RNCE schemes, key generation outputs an additional trapdoor $z$, ignored by honest parties and used by the simulator. Then, there are two ways to generate a ciphertext: (1) an "honest" ciphertext is computed in the standard way $c \leftarrow \mathsf{RNCE.E}(pk, m)$ (so, as in any encryption scheme, it is a commitment to the message), (2) a "fake" ciphertext is computed (by the simulator) without the message, but with the secret key $sk$ and the trapdoor $z$ as $\tilde{c} \leftarrow \mathsf{RNCE.F}(pk, sk, z)$. Given a fake ciphertext $\tilde{c}$ and any message $m$, one can compute a secret key $\tilde{sk} \leftarrow \mathsf{RNCE.R}(pk, sk, z, \tilde{c}, m)$ that explains the message-ciphertext pair (such that $\mathsf{RNCE.D}(\tilde{sk}, \tilde{c}) = m$). Moreover, the distributions $(c, sk)$ (as in the real world) and $(\tilde{c}, \tilde{sk})$ (as in the simulation) are indistinguishable. This allows to explain a single ciphertext per public key.

*The Scheme.* At a high level, the authors of [5] use RNCE to construct non-committing forward-secure public-key encryption by encrypting with a standard forward-secure public-key scheme RNCE ciphertexts instead of messages. We generalize this idea (and the simulation technique) to SM protocols. In particular, we can construct an ASM scheme by taking a standard SM scheme that suffers from the commitment problem and sending, instead of messages, their RNCE encryptions, where each message is encrypted with a different public key. When a message is received, the secret key is immediately deleted. (For the moment, assume that whenever Alice sends a message, an RNCE key pair is "magically" generated—Alice uses the public key, and the secret key immediately appears stored in Bob's state.) This way, the modified scheme inherits all guarantees of the original SM scheme. Furthermore, it can be simulated in the adaptive setting, as we will see below.

Let us now address the problem of how the RNCE keys are distributed. One trivial solution would be to include $\ell$ key pairs as part of the setup: the parties send their $\ell$ public keys at the beginning over an authenticated channel. First, this way we can send only $\ell$ messages overall. But even worse, the RNCE keys do not heal: when the receiver is corrupted for the first time, the simulator can explain all messages sent so far, but it also has to commit to all RNCE secret keys. Hence, adaptive security is never restored. To deal with this, we use the technique used in all SM schemes: we send with each message an update, consisting of $\ell$ fresh RNCE public keys. In particular, Alice (Bob will proceed

analogously) stores some public keys previously received from Bob. When she sends the $i$-th message, she RNCE-encrypts it with one of the unused public keys, generates $\ell$ new key pairs, stores the secret keys, and sends the RNCE ciphertext, the $\ell$ public keys and $i$ to Bob over the channel constructed by an SM scheme. Bob stores the greatest index $i$ he has seen so far. Whenever he sees a message with a greater $i$, he ignores all RNCE public keys he has and replaces them by the $\ell$ newly received ones. Unlike in the first trivial solution, in the above protocol adaptive security is restored as fast as possible: with the first new message delivered from the other party.

*Simulation.* We give an intuition of how the above protocol can be simulated. Assume that the SM scheme has the standard simulator, as hard-coded in most game-based definitions. In particular, he executes the protocol, and when a memory is exposed, he shows to the distinguisher the real state. For ciphertexts corresponding to confidential messages it shows encryptions of 0's, while for non-confidential ones it shows encryptions of the actual message.

In the adaptive setting, the real and the ideal world are easily distinguishable for that simulator. This is because when a message is sent as confidential, and later the memory is exposed, the distinguisher sees in the ideal world the encryption of 0's. However, we can fix this with our new scheme: the new simulator encrypts, instead of 0's, a fake RNCE ciphertext to generate a ciphertext corresponding to a confidential message. When a memory is corrupted, he receives the message (which, of course, can no longer be confidential) and computes the fake RNCE secret key according to the fake ciphertext. RNCE guarantees that this is indistinguishable from the real world, where we have honest ciphertext and an honest key.

*A Note on Efficiency.* First, observe that using a symmetric non-committing encryption scheme, such as the one-time pad, instead of RNCE would not work. This is because in many SM schemes corrupting the sender has no effect on confidentiality, implying that upon such a corruption, the simulator needs to output a key of the symmetric non-committing scheme without knowing the messages (which trivially breaks against a distinguisher knowing the message).

Moreover, while our construction of using nested encryption appears to be redundant, it can be observed that using RNCE only would not suffice. This is because SM schemes can provide certain advanced *confidentiality* guarantees not achieved by RNCE alone. For example, the optimal schemes such as [9,24] provide so-called post-impersonation guarantees: once the adversary injects a message to Bob (after corrupting Alice) and then corrupts Bob, all messages sent by Alice afterwards are confidential.

*Limitations.* Our protocol requires a fixed upper bound on the number of messages a party can send without interaction (in particular, after $\ell$ messages it needs a new set of public keys from the partner). Unfortunately, overcoming this seems unlikely with our approach. This is due to the impossibility result

by Nielsen [20]. It essentially says that a non-committing non-interactive public-key encryption scheme requires that the length of a secret key is at least the overall length of all messages encrypted. This means that we would need non-committing encryption, where the public and secret keys are updated, in other words, a non-committing equivalent of HIBE. To the best of our knowledge, this does not exist yet.[8]

### 6.2 The Construction: Combining RNCE with HIBE

Recall that the HIBE protocol from Sect. 5.2 is designed for the sesqui-directional setting, where it protects the confidentiality of messages sent by Alice. In the protocol, Bob sends to Alice HIBE master public keys, which results in epochs. In epoch $j$, Alice uses the $j$-th master public key to encrypt her messages with the transcript as identity. In this section we consider the analogous setting for the ASM protocol, consisting of RNCE composed with HIBE. That is, Bob sends $\ell$ RNCE keys alongside the HIBE keys, and Alice uses them to additionally encrypt her messages.

Hence, for the ASM construction we need in the real world the additional randomness $\mathsf{Rand}^{renc_i,\mathtt{A}}$ for RNCE-encrypting the $i$-th message and $\mathsf{Rand}^{rkg_j,\mathtt{B}}$ for generating the $j$-th set of $\ell$ keys, compared to the real world from the HIBE protocol. Moreover, we have memories $\mathsf{Mem}^{rsk_{(j,k)},\mathtt{B}}$ for storing the $k$-th RNCE secret key, generated in epoch $j$, and insecure (rewritable) memories $\mathsf{IMem}^{rpk,\mathtt{A}}$ for storing the set of RNCE public keys. Overall, the real-world resources are as follows.

$$\mathsf{R}^{\mathsf{ad\text{-}hibe}} := \Big[\mathsf{R}^{\mathsf{hibe}}, \big\{\mathsf{Rand}^{renc_i,\mathtt{A}}\big\}_{i\in[n]}, \big\{\mathsf{Rand}^{rkg_j,\mathtt{B}}\big\}_{j\in[n]}, \mathsf{IMem}^{rpk,\mathtt{A}}, \big\{\mathsf{Mem}^{rsk_{(j,k)},\mathtt{B}}\big\}_{j\in[n],k\in[\ell]}\Big], \quad (10)$$

where $\mathsf{R}^{\mathsf{hibe}}$ should be understood as the same set of resources as in Sect. 5.2. The restrictions on those set of resources are dropped, on the other hand, since we no longer need work around the commitment problem. This implies, however, that we have to directly consider security of the overall compiled protocol, instead of using the construction statement for HIBE and composition.[9] A formal description of the converters $\mathsf{rnce\text{-}enc}$ and $\mathsf{rnce\text{-}dec}$ implementing the RNCE protocol on top of the HIBE protocol is given in Fig. 7.

In the ideal world, we have the same $2n$ channels: $\mathsf{S}^{\mathsf{ad\text{-}hibe}} := \mathsf{S}^{\mathsf{hibe}}$. Most properties of the constructed channels are the same as in the HIBE construction. In fact, our adaptive protocol only affects (1) availability—only $\ell$ messages can

[8] Note that the impossibility of [20] also rules out a solution where Alice RNCE-encrypts for Bob a new RNCE secret key, used for the next message—this secret key would leave no space for the message.

[9] In general, the simulator for the SM scheme simply does not output the secret state from the commitment-causing memories, and our ASM simulator cannot generate it himself, since this would be inconsistent with the rest of the SM simulation.

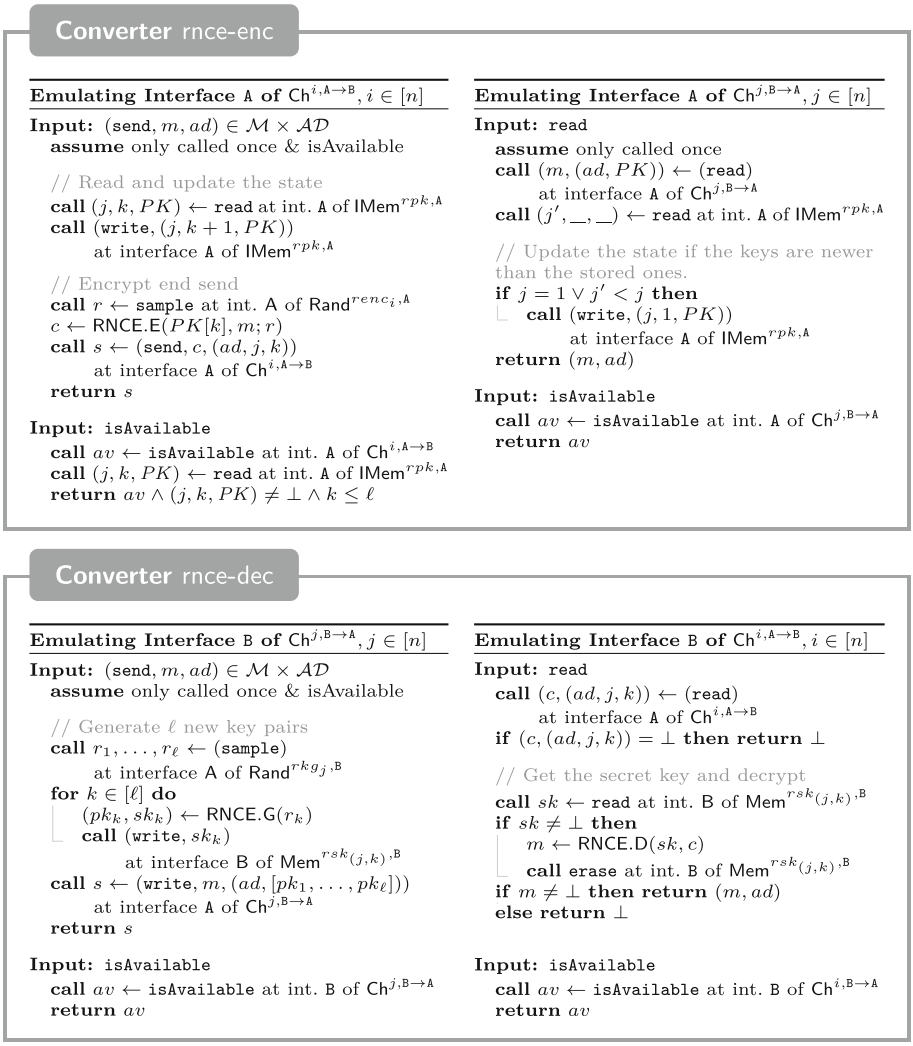


**Fig. 7.** The RNCE part of the adaptively-secure protocol in the sesqui-directional setting.

be sent without interaction, and (2) confidentiality—we need to account for the additional randomness and memory resources. Recall that the epoch $j_i$ in which message $i$ is sent by Alice is determined by the sent and received events. With this, the restriction (1) can be expressed with the can-send and can-receive predicate in a straightforward way.

Let us now focus on confidentiality. Recall that in the HIBE protocol, the can-leak predicate was defined using the event $\mathcal{E}_i^{\mathsf{exposed}}$, denoting that the $i$-th message sent by Alice is inherently insecure. We modify this event to account for

the additional resources used by RNCE. Specifically, the message is exposed if the RNCE-encryption randomness leaks: $\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(renc_i,\mathtt{A})}$, or if the RNCE secret key leaks. The latter happens if Bob's key-generation randomness leaks: $\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(rkg_{j_i},\mathtt{B})}$, or if the secret key memory leaks: $\mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(rsk_{(j_i,k_i)},\mathtt{B})}$, where the $i$-th message was the $k_i$-th one sent in its epoch. Overall, this leads to the following composed event:

$$\mathcal{E}_i^{\mathsf{exposed\text{-}ad}} := \mathcal{E}_i^{\mathsf{exposed}} \vee \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(renc_i,\mathtt{A})} \vee \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Rnd}(rkg_{j_i},\mathtt{B})} \vee \mathcal{E}^{\mathsf{leaked}}_{\mathsf{Mem}(rsk_{(j_i,k_i)},\mathtt{B})}.$$

The leakage function $\mathfrak{L}^{\mathsf{S}^{\mathsf{ad\text{-}hibe}}}_{\mathsf{Ch}(\mathtt{A}\to\mathtt{B})}$ is then defined analogously to that of the HIBE construction `silent` in case of $\mathcal{E}_i^{\mathsf{exposed\text{-}ad}}$, and `false` otherwise. We stress that the need to include these additional cases only arises from our fine-grained modeling of memory and randomness. In reality, it makes sense to consider only one memory storing the whole secret state, only one randomness for RNCE and HIBE encryption, and so on. In such a model, the confidentiality of our adaptively secure scheme and the non-adaptive one would coincide.

The security of our composed protocol is summarized in the following theorem. The proof can be found in the full version [11].

**Theorem 3.** *Let* $\mathsf{R}^{\mathsf{ad\text{-}hibe}}$ *be as in* (10)*, and let* $\mathsf{S}^{\mathsf{ad\text{-}hibe}}$ *be as in above with the described confidentiality guarantees, in-order sending and receiving, and the restriction to* $\ell$ *messages per epoch. If the HIBE scheme is IND-CCA secure with our additional assumptions, then there exists an efficient simulator* $\mathsf{sim}$*, such that*

$$\mathsf{rnce\text{-}enc}^{\mathtt{A}}\mathsf{hibe\text{-}enc}^{\mathtt{A}}\ \mathsf{rnce\text{-}dec}^{\mathtt{B}}\mathsf{hibe\text{-}dec}^{\mathtt{B}}\ \mathsf{R}^{\mathsf{ad\text{-}hibe}} \ \hat{\approx}_{\tau}\ \ \mathsf{sim}^{\mathtt{E}}\ \mathsf{S}^{\mathsf{ad\text{-}hibe}},$$

*where* $\tau$ *is the same event mapping as in Theorem 2.*

## References


1. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: security notions, proofs, and modularization for the signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 129–158. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_5
2. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted encryption and key exchange: the security of messaging. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 619–650. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_21
3. Broadnax, B., Döttling, N., Hartung, G., Müller-Quade, J., Nagel, M.: Concurrently composable security with shielded super-polynomial simulators. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 351–381. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_13
4. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd IEEE Symposium on Foundations of Computer Science - FOCS 2001, pp. 136–145. IEEE Computer Society (2001)

5. Canetti, R., Halevi, S., Katz, J.: Adaptively-secure, non-interactive public-key encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 150–168. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_9
6. Canetti, R., Krawczyk, H.: Universally composable notions of key exchange and secure channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_22
7. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A formal security analysis of the signal messaging protocol. In: 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017, pp. 451–466 (2017)
8. Durak, F.B., Vaudenay, S.: Bidirectional asynchronous ratcheted key agreement with linear complexity. Cryptology ePrint Archive, Report 2018/889 (2018). https://eprint.iacr.org/2018/889
9. Jaeger, J., Stepanovs, I.: Optimal channel security against fine-grained state compromise: the safety of messaging. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_2
10. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 159–188. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_6
11. Jost, D., Maurer, U., Marta, M.: A unified and composable take on ratcheting. Cryptology ePrint Archive, Report 2019/694 (2019). https://eprint.iacr.org/2019/694
12. Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: (De-)constructing TLS 1.3. In: Biryukov, A., Goyal, V. (eds.) INDOCRYPT 2015. LNCS, vol. 9462, pp. 85–102. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26617-6_5
13. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How Secure Is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_19
14. Kuesters, R., Tuengerthal, M., Rausch, D.: The IITM model: a simple and expressive model for universal composability. Cryptology ePrint Archive, Report 2013/025 (2013). https://eprint.iacr.org/2013/025
15. Maurer, U.: Constructive cryptography – a new paradigm for security definitions and proofs. In: Mödersheim, S., Palamidessi, C. (eds.) TOSCA 2011. LNCS, vol. 6993, pp. 33–56. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27375-9_3
16. Maurer, U.: Indistinguishability of random systems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 110–132. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_8
17. Maurer, U., Pietrzak, K., Renner, R.: Indistinguishability amplification. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 130–149. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_8
18. Maurer, U., Renner, R.: Abstract cryptography. In: Innovations in Computer Science - ICS 2011, pp. 1–21. Tsinghua University (2011)
19. Maurer, U., Renner, R.: From indifferentiability to constructive cryptography (and back). In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 3–24. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_1

20. Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_8
21. Open Whisper Systems. Signal protocol library for Java/Android. GitHub repository (2017). https://github.com/WhisperSystems/libsignal-protocol-java. Accessed 01 Oct 2018
22. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_10
23. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Proceedings 2001 IEEE Symposium on Security and Privacy - S&P 2001, pp. 184–200, May 2001. https://doi.org/10.1109/SECPRI.2001.924298
24. Poettering, B., Rösler, P.: Towards bidirectional ratcheted key exchange. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 3–32. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_1
25. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing, STOC 2004, pp. 242–251. ACM, New York (2004). https://doi.org/10.1145/1007352.1007394. http://doi.acm.org/10.1145/1007352.1007394