Secure Channels and Termination: The Last Word on TLS

Colin Boyd Britta Hale

NTNU, Norwegian University of Science and Technology, Trondheim, Norway
{colin.boyd,britta.hale}@item.ntnu.no

Abstract. Secure channels are one of the most pivotal building blocks of cryptography today. Internet connections, secure messaging, protected IoT data, etc., all rely upon the security of the underlying channel. In this work we define channel protocols, as well as security for channels constructed from stateful length-hiding authenticated encryption (stLHAE) schemes. Furthermore, we initiate the concept of *secure termination* where, upon receipt of a signifying message, a receiver is guaranteed to have received every message that has been sent, and will ever be sent, on the channel. We apply our results to real-world protocols, linking the channel environment to previous analyses of TLS 1.2, and demonstrating that TLS 1.2 achieves secure termination via fatal alerts and `close_notify` messages, per the specification of the Alert Protocol.

Keywords: Secure channels, stateful length-hiding authenticated encryption (stLHAE), authenticated encryption with associated data (AEAD), secure termination, controllable channel protocol, Transport Layer Security (TLS)

1 Introduction

Communication security is built on a fundamental cornerstone commonly referred to as a *secure channel*. Creation of secure channels is the essential goal of secure email, end-to-end encrypted messaging applications, end-to-end encrypted VOIP, HTTPS internet connections and TLS in general, WPA2 WiFi protection, SSH, IPSec, Bluetooth, etc. Examples are innumerable. Additionally, many constructs rely on the existence of secure channels once established, e.g. key transport. Despite this, a general understanding of what *secure channels* are and how they are constructed is lacking. Research relating to secure channels has spiraled concentrically around the topic with frequently contradicting goals, particularly in the analysis of real-world protocols.

Authenticated encryption with associated data (AEAD), or even simply authenticated encryption (AE), has been argued as the foundational secure channel building block. Extensive work has been done on both AEAD [32], and AE in their various forms [3, 17, 21, 34, 16, 33]. Stateful length-hiding AE (stLHAE) is often the apparent goal of real-world protocols and has consequently been used frequently in their analysis [20, 23, 29]. Work has also been undertaken for building secure channels explicitly from AE schemes [28]. However, the view of secure channels as simply AEAD or AE is incomplete. In real-world protocols, multiple instances of a protocol may be run, session resumption/renegotiation may be

performed, and message authentication (MAC), encryption, and even exporter keys may be derived from a single master session key output of a key exchange protocol. Essentially, the real-world is not simple. Cross protocol attacks, renegotiation attacks [13], and triple handshake attacks [5] are just some attack examples that cannot be captured when considering secure channels as AEAD or AE under a single communication flow.

Expanding on the modeling of channels, authenticated and secure channel establishment (ACCE) was proposed, which considers both key exchange and channels established under the derived key in the context of stLHAE [20]. Furthermore, ACCE considers parallel sessions – an improvement over the basic, low-level view of secure channels as the stLHAE primitive. Work analyzing the TLS protocol [20, 23] has employed ACCE as the foundational secure channel building block, as TLS does not achieve key indistinguishability. However, not all protocols suffer from a lack of key indistinguishability (e.g. the current TLS 1.3 draft [30]), thus rendering the ACCE pre-accept/post-accept phase combination unnecessary. Similarly to ACCE, Augmented Secure Channels (ASC) have been proposed as a means of capturing more of the channel context than an AEAD primitive allows [2]. ASC is developed in the vein of constructive cryptography, but still suffers from the same underlying drawbacks as ACCE; namely the inability to model communication flows, under potentially different security demands, which are protected by keying material derived from the same master secret. Neither ACCE nor ASC model session resumption – despite its importance in TLS 1.2 (analyzed in the ACCE model) and TLS 1.3 (analyzed in the ASC model for draft 8). Work has also been done on multi-key channels [14], focusing on the evolution of a master key over time, and requirements on it (e.g. forward secrecy), but lacking formal definitions for the channel context. It also uses a single, fixed AEAD construction, lacking flexibility in scheme selection, and does not address termination in the channel environment.

In response to these issues, we define keyed two-party controllable channel protocols which capture parallel and consecutive sessions, with security controlled via the underlying primitives. Each session at a principal is initiated via the generation of a master session key – as would be the case at the conclusion of a key exchange protocol. In turn, the session is modeled by a collection of read and write connections, with connection keys derived from the master session key. Channels are defined by a shared key between a pair of read and write connections. This captures the behavior of real-world protocols: for example, a master session key may be used to derive a MAC key as well as an encryption key, which would result in parallel connections under the different channel keys. Simultaneously, this higher-level view of the channel environment provides a framework for analysis of session resumption; connections can be closed, and channel keys destroyed, with new connections instantiated and channel keys derived from the original session keying material. Compared to the ACCE model, our model permits separate consideration of session and connection keys and even consideration of connection closure.

Security for keyed two-party controllable channel protocols is realized via the cryptographic schemes and key derivation functions (KDF) used for connections. In real-world correlations, TLS 1.2 uses cipher suites – for example, the partial suite AES_128_GCM_SHA256, which defines both a scheme for securing the channel (AES_128_GCM) and pseudo-random function for key derivation of the channel key (HMAC using SHA256). Our model captures such real-world protocols by considering channel security under both a scheme (e.g. stLHAE) goal and a KDF goal. Considering secure channels in the context of stLHAE allows certain analysis benefits; statefulness gives assurance that the i-th packet processed on the receiving end of a channel is the same as i-th packet output on the sending end. However, the stLHAE framework is again incomplete. How does a receiver know that all sent packets have been received? If an adversary drops the last n packets on a channel, the receiver could convinced that the transmission is shorter than in reality. When final messages contain critical information, warnings, etc., this scenario should undoubtedly be considered as an attack. Classically, this is referred to as a *truncation* attack. In order to capture this attack in our secure channel environment, we define secure termination.

Truncation attacks have been shown to be a very real problem [35], including effects on voting. Usually these attacks follow from a failure of the implementation to check for the closure alert (if TLS), or from a misinterpretation of what constitutes a "terminate" message. While the importance of the former is highlighted by our model, the latter is at the heart of secure termination. Most analyses of truncation attacks are ad-hoc, essentially cryptanalyses based on weaknesses discovered in a particular protocol. Some of these have been against the TLS protocol [27, 35], leveraging and exploiting implementation faults. Other recent work on protocol termination, albeit in an unrelated aspect, highlights a growing interest in the final stages of a cryptographic protocol run [10]. Consequently, we define secure termination in the interest of providing a formal framework for modeling truncation attacks, by modeling finalization and completion guarantees on received communication flows.

Channels To discuss secure channels and channel termination it is vital to clearly define what a channel actually is. Past work using channels has provided mixed descriptions of this concept – Hoepman [18] describes *unidirectional* and *bidirectional* channels, hence conceptually equating a channel to a transport link between entities. In this sense, a bidirectional channel for a real-world protocol such as TLS would have separate keys for each direction, but all keys would be considered to be within the same channel. Meanwhile, another line of research [26, 25] defines a channel in a unidirectional sense, with messages input from a sender and output to a receiver. Ultimately, this formulation allows for a modular analysis of channel security, with channel keys being used for sending messages at one end of the channel and for receiving messages at the other end. This practice has met with wide-spread acceptance, with channels generally being modeled by three interfaces (sender, receiver, and adversary) based on various adversarial capabilities [2].

In 2001, Canetti and Krawczyk [9] defined a *secure channel* as "a link between a pair of parties" which provides message authentication and confidentiality via a key obtained via a key-exchange protocol. As in many other works on key exchange and secure channel analysis [20, 23, 4], the authors consider *sessions* at principals as participants in the key exchange protocol, and call the resulting key the *session key*. This session key is then used to secure the channel (e.g. encryption and authentication keys are derived from the session key).

One salient issue arises when the standard conceptualization of a session key is compared with the modeling of a channel, as discussed above. Namely, a key exchange protocol should minimally result in a set of two channel keys. This assumption follows from standard real-world protocols which maintain separate session keys for sending channels and receiving channels [11, 19, 31]. In consideration of this, we undertake to formalize channels in the context of sessions for bi-directional communication. With sent messages at a session being not necessarily independent from received messages (particularly, as we will see, in the context of secure termination), this new, "big picture" view of session modeling raises interesting questions for channel analysis. One recent work [24] similarly aims to address bi-directional channels, but does not consider sub-connections (parallel channels after key derivation) or channel termination.

While TLS and similar protocols have been analyzed under the assumption of discrete messages, important work has been done considering AEAD in the streaming setting [12]. We view our work as easily adaptable to the streaming security context, since the security of our channel protocol model is reliant upon underlying, "plug-in" primitives and their security games.

Cryptographic Agility While channels generally use separate keys for sending and receiving, these keys are often generated from the same master session key. This is indeed the case in many real-world protocols such as TLS. Moreover, it is possible that the same master session key could be re-used with different concrete cryptographic algorithms, which may be controllable by the adversary. We account for this possibility by requiring *agile* [1, 15] channel key derivation in our channel security analysis. Bhargavan et al. [6] analyzed cryptographic agility in TLS, focusing on the public key algorithms used in the handshake protocol. Agility definitions appear in the full version due to space restrictions. We apply agile key derivation functions in our results in order to provide strong channel security.

Paper Outline Introducing keyed two-party controllable channel protocols, §2 defines sessions, connections, schemes, channels, and connection/channel closure, outlining the fundamental structure of the channel environment even before consideration of security. Combinations of security schemes and key derivation functions (for the channel key) comprise *suites*.

Subsequently, §3 describes security. We focus on channel stLHAE security due to it being a frequent real-world objective. Our security experiments here are broad, and encompass the situation where parallel and consecutive connections may employ various suites. We expect that this work can be extended to

situations where parallel and consecutive connections may employ combinations of various authentication suites, stLHAE suites, AEAD suites, etc.

Additionally, we provide constructions of channel protocol environments where channels are constructed from various stLHAE schemes. Proving security for our constructions, we provide a reduction from the channel security to the security of the underlying schemes used in the construction.

Linking our definitions and constructions to the ACCE model, we furthermore demonstrate that channel stLHAE security can be reduced to ACCE security in the post-accept phase (channel phase of ACCE), when the session key is used directly as the channel key and only one connection for the session is initialized (§5.1). Thus our channel protocol can be used as a refined post-accept phase for the ACCE model, if such a model is necessary due to lack of key indistinguishability. Formally, the one-connection and session key restrictions on the reduction arise from the formalization of the ACCE model, as keyed two-party controllable channel protocols encompass a wider vision of protocol interaction, based on real-world protocol implementations.

In §4 secure termination and the secure termination experiment are defined. Additionally, we associate secure termination and channel protocol security by reductions between the former and the latter, under the generic case of parallel and consecutive connections constructed from various stLHAE suites. We demonstrate that TLS 1.2 achieves secure termination on receipt of Alert or `close_notify` messages by combining our work on secure channels with previous analyses of the TLS 1.2 protocol in the ACCE model.

While presented in detail in the following sections, the notation for channel protocols, schemes, and respective security games is summarized in Table 1. This provides a reference point, highlighting the differences between similar terms and notation, and is an extension of previously established notation.

ChannelSnd ChannelRcv	Send and receive algorithms for a keyed two-party controllable channel protocol $\textsc{Chnl}[\{(\Pi, \text{KDF})\}]$.
$\mathsf{ChnlEnc}$ $\mathsf{ChnlDec}$	Security experiment oracles for a keyed two-party controllable channel protocol $\textsc{Chnl}[\{(\Pi, \text{KDF})\}]$, where $\{\Pi\}$ are stLHAE schemes.
$\mathsf{Send}_{\mathsf{T}}$ $\mathsf{Receive}_{\mathsf{T}}$	Security experiment oracles for the secure termination of a channel protocol experiment.
Enc & Dec	Encrypt and decrypt algorithms for a stLHAE scheme.

Table 1. Notation reference for protocols, schemes, and experiments.

2 Channels

In this section we formalize the natural real-world protocol environment of sessions, connections, and channels. Fig. 1 provides a depiction of this environment.

2.1 Definitions

Every channel protocol will be associated with one or more cryptographic *schemes*. Typical schemes include authentication schemes, under a message authentication code, and stLHAE, but many other types are possible. We define abstractly the elements of any such scheme.

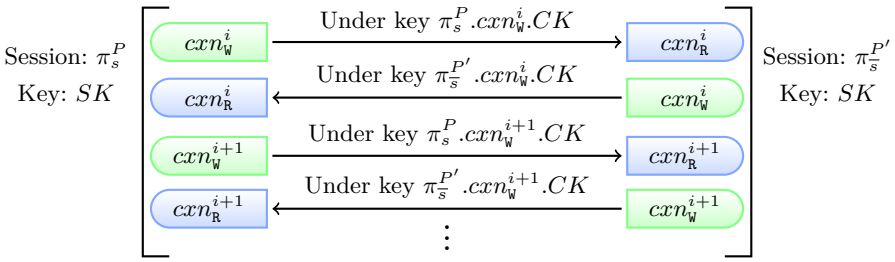

Fig. 1. Communication diagram with sessions π_s^P and $\pi_{\overline{s}}^{P'}$, write and read connections $cxn_{\mathtt{W}}$ and $cxn_{\mathtt{R}}$ for each session, and channels between connections protected by the channel keys $\pi_s^P.cxn_{\mathtt{W}}^i.CK$, etc. Each principal may be modeled by multiple sessions.

Definition 1 (Scheme). *Let $\mathcal{M}$ be a message space, $\mathcal{K}$ a key space, and $\mathcal{C}$ an output space, and let the elements of $\mathcal{C}$ be called ciphertexts. A* scheme *Π is a tuple of algorithms:*

- $\mathrm{Kgn}() \xrightarrow{\$} k$*: A probabilistic key gen. algorithm that outputs a key $k \in \mathcal{K}$.*
- $\mathrm{Snd}(k, m, st_{\mathrm{S}}) \xrightarrow{\$} (c, st'_{\mathrm{S}})$*: A probabilistic send algorithm that takes as input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a write state st_{S}, and outputs an outgoing ciphertext $c \in \mathcal{C}$ or an error symbol $\perp$, and an updated state st'_{S}.*
- $\mathrm{Rcv}(k, c, st_{\mathrm{R}}) \to (m, st'_{\mathrm{R}})$*: A deterministic receive algorithm that takes as input a key $k \in \mathcal{K}$, a ciphertext $c \in \mathcal{C}$, and a read state st_{R}, and outputs either a message $m \in \mathcal{M}$ or a error symbol $\perp$, and an updated state st'_{R}.*

On first use, st_{S} and st_{R} are initialized to st^0_{S} and st^0_{R}, resp. If, for $\mathrm{Snd}(k, m, st_{\mathrm{S}}) \xrightarrow{\$} (c, st'_{\mathrm{S}})$ and $\mathrm{Rcv}(k, c, st_{\mathrm{R}}) \to (m, \alpha, st'_{\mathrm{R}})$, $st'_{\mathrm{S}} = \perp$ and $st'_{\mathrm{R}} = \perp$, then Π is said to be stateless. *Otherwise Π is said to be* stateful.

Correctness Consider the following: $i \geq 0$, all $m_i \in \mathcal{M}$, all $k \xleftarrow{\$} \mathrm{Kgn}()$, initial states st^0_{S} and st^0_{R}, and a sequence $(c_i, st^{i+1}_{\mathrm{S}}) \xleftarrow{\$} \mathrm{Snd}(k, m_i, st^i_{\mathrm{S}})$, where $c_i \neq \perp$ for all i. Then, for a matching sequence of message receipts, we have $(m_i, st^{i+1}_{\mathrm{R}}) \leftarrow \mathrm{Rcv}(k, c_i, st^i_{\mathrm{R}})$. Further correctness requirements may hold, dependent on the scheme. In §3, stLHAE schemes will be considered; however, many scheme types are possible. Other types of schemes include signatures schemes and authentication schemes.

Definition 2 (Principals and Sessions). *For a collection of principals $\{P_1, \ldots, P_p\}$, where $p \in \mathbb{N}$, each $P_l \in \{P_1, \ldots, P_p\}$ is (potentially) in possession of a long-term private/public key pair (sk_i, pk_i), and is modeled by a collection of* session *oracles $\{\pi^{P_l}_{s_1}, \ldots, \pi^{P_l}_{s_n}\}$.*

Correspondingly and wlog, each session π_s^P is an oracle with access to the (potential) long-term key pair of P and the (potential) long-term public keys $pk_1, \ldots, pk_p$ of all other principals. Furthermore, a session *π_s^P at a principal P maintains a collection of variables $(\{\pi_s^P.cxn^i_{role}\}, \pi_s^P.(P', \overline{s}), \pi_s^P.\alpha, \pi_s^P.SK)$:*

- *$\{\pi_s^P.cxn^i_{role}\}$: A collection of connections, where $role \in \{\mathtt{W}, \mathtt{R}\}$, representing 'write' and 'read' connections, respectively.*
  If $role = \mathtt{W}$, then $\overline{role}$ denotes $\mathtt{R}$, and vice versa.

– $\pi_s^P.(P', \overline{s})$*: An identifier for the partner's session.*
– $\pi_s^P.\alpha$*: A status variable in* $\{0, 1\}$*, where the session is active if* $\alpha = 1$ *and inactive if* $\alpha = 0$*. A session must be active to send or receive messages.*
– $\pi_s^P.SK$*: A session key shared with the partner session.*

Any P *may maintain several sessions, both in parallel and consecutively.*

The following definition uses a key derivation function (KDF). We follow standard assumptions by requiring KDF to be a PRF [8]; KDFs may also be defined more explicitly (see [22]).

Definition 3 (Connection). *A* connection cxn_{role} *with role* $\in \{\mathtt{W}, \mathtt{R}\}$ *is defined by a set of variables* $(cxn_{role}.CK, cxn_{role}.status, cxn_{role}.suite, cxn_{role}.substate)$*:*

– $cxn_{role}.CK$*: A channel key corresponding to the connection and role.*
– $cxn_{role}.status$*: A status variable in* $\{\mathtt{active}, \mathtt{terminated}\}$*.*
– $cxn_{role}.suite$*: A variable identifying the scheme/KDF pair* (Π, KDF) *implemented on the connection. If no pair is specified for the connection, then* $cxn_{role}.suite \leftarrow \perp$*.*
– $cxn_{role}.substate$*: Any additional scheme-specific connection state variables.*

The connection *substate* variable models other state information, which may be defined by *suite*. For example, this variable may handle protocol state if Π is a stLHAE scheme. Exact use of this variable can be seen in the concrete constructions provided in §3. We abuse notation and use cxn_{role} as an identifier by which an entity may refer to the collection of variables, without having access to them.

Definition 4 (Keyed Two-Party Controllable Channel Protocol). *Let* $\mathcal{M} = \mathcal{AD} \times \mathcal{AED}$ *be a message space and* $\mathcal{L}$ *an optional length space. A keyed two-party controllable channel protocol* $\textsc{Chnl}[\{(\Pi, \mathrm{KDF})\}]$ *over a set of scheme, key derivation function pairs* $\{(\Pi, \mathrm{KDF})\}$ *is a tuple of algorithms:*

– $\mathrm{SessionKeyGen}(1^\kappa, P, P', s, \overline{s}) \xrightarrow{\$} SK$*: A probabilistic* session key generation algorithm *that takes as input a security parameter* κ*, principals* P *and* P'*, and session indices* s *and* $\overline{s}$*. It sets the respective partner identifiers and session status* $\pi_s^P.\alpha \leftarrow 1$*, and outputs a shared session key* SK*.*
– $\mathrm{ConnectionInit}(SK, \pi_s^P, \pi_{\overline{s}}^{P'}, [\Pi], [\mathrm{KDF}], i) \xrightarrow{\$} (\pi_s^P.cxn_{\mathtt{W}}^i, \pi_s^P.cxn_{\mathtt{R}}^i)$*: A probabilistic* connection initiation algorithm *that takes as input a tuple of a shared session key* SK*, two partner sessions* π_s^P *and* $\pi_{\overline{s}}^{P'}$*, an optional scheme* Π*, an optional key derivation function* KDF *used to derive* CK*, and connection index* i*, and outputs the* i*-th 'read' and 'write' connections at the first input session,* $\pi_s^P.cxn_{\mathtt{W}}^i$ *and* $\pi_s^P.cxn_{\mathtt{R}}^i$ *or a distinguishing failure symbol* $\perp$*.*
– $\mathrm{ChannelSnd}(m, [\ell], cxn_{\mathtt{W}}^i) \xrightarrow{\$} (c, cxn_{\mathtt{W}}^{i\,\prime})$*: A probabilistic* channel sending algorithm *that takes as input a message* $m \in \mathcal{M} = \mathcal{AD} \times \mathcal{AED}$*, an optional output length* $\ell \in \mathcal{L}$*, and a connection* $cxn_{\mathtt{W}}^i$*, and outputs a ciphertext* $c \in \mathcal{C}$ *or a distinguishing failure symbol* $\perp$*, and an updated connection* $cxn_{\mathtt{W}}^{i\,\prime}$*.*

– ChannelRcv$(p, cxn^i_{\mathtt{R}}) \to (m, cxn^{i\,\prime}_{\mathtt{R}})$: *A deterministic* channel receiving algorithm *that takes as input a packet* $p \in \mathcal{P} = \mathcal{AD} \times \mathcal{C}$ *and a connection* $cxn^i_{\mathtt{R}}$*, and outputs a message* $m \in \mathcal{M} = \mathcal{AD} \times \mathcal{AED}$ *or a distinguishing failure symbol* $\perp$*, and an updated connection* $cxn^{i\,\prime}_{\mathtt{R}}$*.*

The packet space $\mathcal{P}$ *is a set induced by* CHNL$[\Pi, \mathrm{KDF}]$ *for each* $\Pi \in \{\Pi\}$*, the length space* $\mathcal{L}$*, and the message space* $\mathcal{M}$*, where* $\mathcal{M}$ *is a tuple of the data space of authenticated transmissions* $\mathcal{AD}$*, and a data space of authenticated and encrypted transmissions* $\mathcal{AED}$*. Consequently,* $\mathcal{P}$ *is a tuple of* $\mathcal{AD}$ *and a ciphertext space* $\mathcal{C}$*, where* $\mathcal{C}$ *is defined by the scheme used on* $\mathcal{M}$*. The i-th read-write connection pair at a session* π^P_s *is denoted* $(\pi^P_s.cxn^i_{\mathtt{W}}, \pi^P_s.cxn^i_{\mathtt{R}})$*.*

We define correctness in the logical way where ConnectionInit outputs the failure symbol if the session is not active, and ChannelSnd and ChannelRcv, respectively, output the failure symbol if the connection status variable is not active. Due to space restrictions, details appear in the full version.

Remark 1. Note that there is no restriction on the number of calls made to ConnectionInit$(SK, \pi^P_s, \pi^{P'}_{\bar{s}}, [\Pi], [\mathrm{KDF}], i)$, for a given session key SK and sessions π^P_s and $\pi^{P'}_{\bar{s}}$. This models real-world protocols where session resumption is possible. Particularly, connections $\pi^P_s.cxn^i_{role}$ and $\pi^P_s.cxn^i_{\overline{role}}$ may be *terminated*, resulting in the connections being destroyed completely; later resumption based on the session keying material is possible by calling ConnectionInit again.

Remark 2. We explicitly allow ConnectionInit to be used to create connections at only one session. Matching connections can be created by using the algorithm again: ConnectionInit$(SK, \pi^{P'}_{\bar{s}}, \pi^P_s, [\Pi], [\mathrm{KDF}], i)$. This matches real-world protocols where each session derives its connection keys independently, regardless of whether or not the partner session is still active.

CHNL may take as input several schemes which may each be used to initialize different connections. If no KDF is used to derive channel keys, then either the session key is used directly in the channel, or no security scheme Π is implemented (i.e. the $([], [])$ suite). In either case $[]$ is included in the list of functions $\{\mathrm{KDF}\}$, and represents the identity function.

Definition 5 (Channel). *If* CHNL *is a keyed two-party controllable channel protocol, and* $\pi^P_s.cxn^i_{\mathtt{W}}$ *and* $\pi^{P'}_{\bar{s}}.cxn^j_{\mathtt{R}}$

– *share a channel key* $\pi^P_s.cxn^i_{\mathtt{W}}.CK = \pi^{P'}_{\bar{s}}.cxn^j_{\mathtt{R}}.CK$ *and*
– *implement the same scheme/*KDF *suite such that* $\pi^P_s.cxn^i_{\mathtt{W}}.suite = \pi^{P'}_{\bar{s}}.cxn^j_{\mathtt{R}}.suite$*,*

then we say that $\pi^P_s.cxn^i_{\mathtt{W}}$ has a channel to $\pi^{P'}_{\bar{s}}.cxn^j_{\mathtt{R}}$*. Moreover, if* $\pi^P_s.cxn^i_{\mathtt{W}}.suite = (\Pi, \mathrm{KDF}) = \pi^{P'}_{\bar{s}}.cxn^j_{\mathtt{R}}.suite$*, then we say that* $\pi^P_s.cxn^i_{\mathtt{W}}$ has a Π-channel to $\pi^{P'}_{\bar{s}}.cxn^j_{\mathtt{R}}$*.*

By not demanding that $i = j$ in Def. 5, we enable modeling of unknown key share. Namely, *any* two connections using the same channel key and suite share a channel, regardless if they were correctly initiated via ConnectionInit.

3 Keyed Two-Party stLHAE Channel Protocol Security

As discussed in the introduction and §2, channel protocols may take as input Π-schemes of various types as well as various KDF-functions. In this section, we introduce stateful length-hiding authenticated encryption (stLHAE) channel security and provide concrete constructions of channels where the set $\{\Pi\}$ is comprised of stLHAE schemes (see App. A).

While the following definition, Def. 6, is not necessary for defining the channel protocol environment, it is essential for consideration of its security. Termination messages affect connection status, and therefore the ability to send or receive messages. Consequently, an adversary could use the encryption of such messages to distinguish between ciphertexts (see Fig. 2).

Definition 6 (Terminate Message). *Let* CHNL *be a keyed two-party controllable channel protocol, and let $\mathcal{T} \subset \mathcal{M}$, the message space of* CHNL*. Then, for $m \in \mathcal{T}$, m is called a* `terminate` *message and, for all $m \in \mathcal{T}$, we add the following to the* correctness *requirements of* CHNL*:*

- *if* $(c, \pi_s^P.cxn_{\mathtt{W}}^{i\,\prime}) \leftarrow \text{ChannelSnd}(m, [\ell], \pi_s^P.cxn_{\mathtt{W}}^i)$ *and* $\pi_s^P.cxn_{\mathtt{W}}^i.status = \mathtt{active}$*, then* $\pi_s^P.cxn_{\mathtt{W}}^{i\,\prime}.status \leftarrow \mathtt{terminated}$*, and*
- *if* $(m, \pi_s^P.cxn_{\mathtt{R}}^{i\,\prime}) \leftarrow \text{ChannelRcv}(p, \pi_s^P.cxn_{\mathtt{R}}^i)$ *and* $\pi_s^P.cxn_{\mathtt{R}}^i.status = \mathtt{active}$*, then* $\pi_s^P.cxn_{\mathtt{R}}^i.status \leftarrow \mathtt{terminated}$*.*

We define *secure stLHAE channels* in Def. 7. Note that we do not present a single, generic *secure channel* definition since security for channel protocols must be considered with respect to the Π-scheme goals (stLHAE, authentication, signatures, etc.). We envisage protocols possibly implementing several Π-schemes. For example, an encryption scheme Π_1 and authentication scheme Π_2. Connections would be initiated using (Π_1, KDF_1), (Π_2,KDF_2), or no security scheme ([][]), denoted $\text{CHNL}[(\Pi_1, \text{KDF}_1), (\Pi_2, \text{KDF}_2), ([], [])]$. In terms of real-world communication, a session could have, and close, connections running HTTPS, while maintaining other connections that send and receive information unprotected via HTTP (i.e. the ([], []) suite). It may then, via a TLS key-renegotiation under the existing session key, initiate new connections for channels that will be protected under an stLHAE Π-scheme. Such possibilities extend the current security considerations and are left for future work.

Definition 7 (stLHAE Channel Security). *Let* $\text{CHNL}[\{(\Pi, \text{KDF})\}]$ *be a keyed two-party controllable channel protocol such that $\{\Pi\}$ are stLHAE schemes, and let $\mathcal{A}$ be a PPT adversarial algorithm. The stLHAE experiment for* $\text{CHNL}[\{(\Pi, \text{KDF})\}]$ *is given by* $\mathsf{Exp}^{\mathsf{stlhae}}_{\text{CHNL}[\{(\Pi,\text{KDF})\}]}$ *in Fig. 2. We define*

$$\mathbf{Adv}^{\mathsf{stlhae}}_{\text{CHNL}[\{(\Pi,\text{KDF})\}]}(\mathcal{A}) = 2\Pr\left[\mathsf{Exp}^{\mathsf{stlhae}}_{\text{CHNL}[\{(\Pi,\text{KDF})\}],\mathcal{A}}(\lambda)\right] - 1\ .$$

An channel protocol $\text{CHNL}[\{(\Pi, \text{KDF})\}]$ *is a* secure channel stLHAE protocol *if* $\mathbf{Adv}^{\mathsf{stlhae}}_{\text{CHNL}[\{(\Pi,\text{KDF})\}]}(\mathcal{A})$ *is a negligible function in λ for all PPT adversaries $\mathcal{A}$.*

In the security game for $\text{CHNL}[\{(\Pi, \text{KDF})\}]$ in Fig. 2, the adversary may select (Π, KDF) pairs used to initiate the channel connections. However, if

$\underline{\mathsf{Exp}^{\mathsf{stlhae}}_{\textsc{Chnl}[\{(\Pi,\mathrm{KDF})\}],\mathcal{A}}(\lambda)}$:

1: $b \xleftarrow{\$} \{0,1\}$, $\mathtt{list} \leftarrow \bot$
2: $b' \leftarrow \mathcal{A}^{\mathsf{SnPairInit}(\cdot),\mathsf{CxnInit}(\cdot),\mathsf{ChnlEnc}(\cdot),\mathsf{ChnlDec}(\cdot)}()$
3: **return** $(b' = b)$

$\underline{\mathsf{SnPairInit}((P,s),(P',\overline{s}))}$:

1: $SK \xleftarrow{\$} \mathrm{SessionKeyGen}(1^\kappa, P, P', s, \overline{s})$
2: **return** $\bot$

$\underline{\mathsf{CxnInit}(\pi_s^P, \pi_{\overline{s}}^{P'}, \Pi^*, \mathrm{KDF}^*, i)}$:

1: **if** $(\Pi^*, \mathrm{KDF}^*) \notin \{(\Pi, \mathrm{KDF})\}$ **then**
2: **return** $\bot$
3: **if** $(\pi_s^P.cxn_{\mathtt{W}}^i \in \mathtt{list}) \vee (\pi_s^P.cxn_{\mathtt{R}}^i \in \mathtt{list})$ **then**
4: **return** $\bot$
5: $(\pi_s^P.cxn_{\mathtt{W}}^i, \pi_s^P.cxn_{\mathtt{R}}^i) \xleftarrow{\$} \mathrm{ConnectionInit}(SK, \pi_s^P, \pi_{\overline{s}}^{P'}, \Pi^*, \mathrm{KDF}^*, i)$
6: $\mathtt{list} \leftarrow \mathtt{list} | \pi_s^P.cxn_{\mathtt{W}}^i | \pi_s^P.cxn_{\mathtt{R}}^i$
7: $u_{(P,s,i)} \leftarrow 0$, $v_{(P,s,i)} \leftarrow 0$, $\mathtt{phase}_{(P,s,i)} \leftarrow 0$
8: **return** $(\pi_s^P.cxn_{\mathtt{W}}^i, \pi_s^P.cxn_{\mathtt{R}}^i)$

$\underline{\mathsf{ChnlEnc}((\mathtt{ad},(m_0,m_1)),[\ell],\pi_s^P.cxn_{\mathtt{W}}^i)}$:

1: $u \leftarrow u + 1$
2: **if** $((\mathtt{ad}, m_0) \in \mathcal{T}) \wedge ((\mathtt{ad}, m_1) \notin \mathcal{T})$ or $((\mathtt{ad}, m_1) \in \mathcal{T}) \wedge ((\mathtt{ad}, m_0) \notin \mathcal{T})$ **then**
3: **return** $(\bot, \pi_s^P.cxn_{\mathtt{W}}^i.status)$
4: $(c^{(0)}, \pi_s^P.cxn_{\mathtt{W}\cdot(0)}^i) \leftarrow \mathrm{ChannelSnd}((\mathtt{ad}, m_0), [\ell], \pi_s^P.cxn_{\mathtt{W}}^i)$
5: $(c^{(1)}, \pi_s^P.cxn_{\mathtt{W}\cdot(1)}^i) \leftarrow \mathrm{ChannelSnd}((\mathtt{ad}, m_1), [\ell], \pi_s^P.cxn_{\mathtt{W}}^i)$
6: **if** $c^{(0)} = \bot$ or $c^{(1)} = \bot$ **then**
7: **return** $(\bot, \pi_s^P.cxn_{\mathtt{W}}^i.status)$
8: $(sent.ad_u, sent.c_u, \pi_s^P.cxn_{\mathtt{W}}^i) \leftarrow (\mathtt{ad}, c^{(b)}, \pi_s^P.cxn_{\mathtt{W}\cdot(b)}^i)$
9: **return** $(sent.c_u, \pi_s^P.cxn_{\mathtt{W}}^i.status)$

$\underline{\mathsf{ChnlDec}((\mathtt{ad},c),\pi_s^P.cxn_{\mathtt{R}}^i)}$:

1: **if** $b = 0$ **then**
2: **return** $(\bot, \pi_s^P.cxn_{\mathtt{R}}^i.status)$
3: $v \leftarrow v + 1$
4: $((\mathtt{ad}, m), \pi_s^P.cxn_{\mathtt{R}}^i) \leftarrow \mathrm{ChannelRcv}((\mathtt{ad}, c), \pi_s^P.cxn_{\mathtt{R}}^i)$
5: **if** $(\exists \pi_{\overline{s}}^{P'}.cxn_{\mathtt{W}}^i$: $\pi_{\overline{s}}^{P'}.cxn_{\mathtt{W}}^i$ has a channel to $\pi_s^P.cxn_{\mathtt{R}}^i$ $) \wedge (\nexists j, j \neq i : \pi_{\overline{s}}^{P'}.cxn_{\mathtt{W}}^j$ has a channel to $\pi_s^P.cxn_{\mathtt{R}}^i)$ **then**
6: $u \leftarrow u_{(P',\overline{s},i)}$
7: **else**
8: $u \leftarrow 0$
9: **if** $(u < v) \vee (c \neq sent.c_v) \vee (\mathtt{ad} \neq sent.ad_v)$ **then**
10: $\mathtt{phase} \leftarrow 1$
11: **if** $\mathtt{phase} = 1$ **then**
12: **return** $((\mathtt{ad}, m), \pi_s^P.cxn_{\mathtt{R}}^i.status)$
13: **return** $(\bot, \pi_s^P.cxn_{\mathtt{R}}^i.status)$

Fig. 2. Oracles of the stLHAE $\textsc{Chnl}[\{(\Pi, \mathrm{KDF})\}]$ security experiment, where Π is specified in $\pi_s^P.cxn_{\mathtt{W}}^i.suite$ (resp. $\pi_s^P.cxn_{\mathtt{R}}^i.suite$). For conciseness, the synchronization variable $u_{(P,s,i)}$ is referenced as u in ChnlEnc, and $v_{(P,s,i)}$ is referenced as v in ChnlDec.

the pair is not valid – in the set $\{(\Pi, \mathrm{KDF})\}$ – no connection will be initiated. ChnlSnd may be called on any connection as ChnlSnd adaptively uses ChannelSnd constructed from the scheme Π from the connection's internal variable $\pi_s^P.cxn_{\mathtt{W}}^i.suite$ (analogously ChnlRcv). For agility between suites, every connection pair is initialized with the same session key SK.

Remark 3. Since multiple Π schemes may be implemented by $\textsc{Chnl}[\{(\Pi, \mathrm{KDF})\}]$, we envisage that it is possible to run various but simultaneous (or consecutive) experiments on the different connections – all linked to the master session key. Note the applications to real-world protocols: in 802.11 [19], TKIP uses a pairwise master key to derive 2 sets of MAC and 2 sets of encryption keys for EAPOL

and application data protection. Different, simultaneous experiments can also be considered for the MAC and encryption goals. This is left for future work.

Definition 8 (Channel Construction from stLHAE). *Let* $\{\Pi\}$*, where* $\Pi = (\mathrm{Kgn}, \mathrm{Enc}, \mathrm{Dec})$*, be stLHAE scheme(s) and let* $\{\mathrm{KDF}\}$ *be key derivation function(s). A keyed two-party controllable channel protocol* $\textsc{Chnl}[\{(\Pi, \mathrm{KDF})\}]$ *is constructed from* Π *and* KDF *to achieve a pair of linked stLHAE channels, with message space* $\mathcal{M} = \mathcal{AD} \times \mathcal{AED}$ *and packet space* $\mathcal{P} = \mathcal{AD} \times \mathcal{C}$*, as follows:*

- $\mathrm{SessionKeyGen}(1^\kappa, P, P', s, \overline{s})$*:*
  - *Selects a shared session key* SK *according to the dist. of key space* $\mathcal{SK}$*,*
  - *sets* $\pi_s^P.\alpha \leftarrow 1$*,* $\pi_{\overline{s}}^{P'}.\alpha \leftarrow 1$*,*
  - *sets respective partner identifiers* $\pi_s^P.(P', \overline{s})$ *and* $\pi_{\overline{s}}^{P'}.(P, s)$*,*
  - *sets* $\pi_s^P.SK \leftarrow SK$ *and* $\pi_{\overline{s}}^{P'}.SK \leftarrow SK$*.*

  Return SK*.*
- $\mathrm{ConnectionInit}(SK, \pi_s^P, \pi_{\overline{s}}^{P'}, \Pi, \mathrm{KDF}, i)$*:*

  If $\mathrm{ConnectionInit}(\cdot, \pi_s^P, \pi_{\overline{s}}^{P'}, \cdot, \cdot, i)$ *has previously been called, return* $\perp$*.*
  - *Compute* $\pi_s^P.cxn_{\mathtt{W}}^i.CK \leftarrow \mathrm{KDF}(SK, (P, s), (P', \overline{s}), i)$ *and*
  - $\pi_s^P.cxn_{\mathtt{R}}^i.CK \leftarrow \mathrm{KDF}(SK, (P', \overline{s}), (P, s), i)$*.*

  If $\pi_s^P.cxn_{\mathtt{W}}^i.CK$ *or* $\pi_s^P.cxn_{\mathtt{R}}^i.CK$ *are not in the key space* $\mathcal{K}$ *of* Π*, return* $\perp$*.*
  - *Set status* $\pi_s^P.cxn_{\mathtt{W}}^i.status = \mathtt{active}$*, and* $\pi_s^P.cxn_{\mathtt{R}}^i.status = \mathtt{active}$*,*
  - *set suite* $\pi_s^P.cxn_{\mathtt{W}}^i.suite \leftarrow (\Pi, \mathrm{KDF})$*, and* $\pi_s^P.cxn_{\mathtt{R}}^i.suite \leftarrow (\Pi, \mathrm{KDF})$*,*
  - $\pi_s^P.cxn_{\mathtt{W}}^i.substate \leftarrow 0$*, and* $\pi_s^P.cxn_{\mathtt{R}}^i.substate \leftarrow 0$*.*

  Return $(\pi_s^P.cxn_{\mathtt{W}}^i, \pi_s^P.cxn_{\mathtt{R}}^i)$*.*
- $\mathrm{ChannelSnd}((\mathtt{ad}, m), \ell, \pi_s^P.cxn_{\mathtt{W}}^i)$*:*

  If $\pi_s^P.cxn_{\mathtt{W}}^i.status \neq \mathtt{active}$*, return* $(\perp, \pi_s^P.cxn_{\mathtt{W}}^i)$*. Otherwise,*
  - *compute* $(c, \pi_s^P.cxn_{\mathtt{W}}^i.substate') \xleftarrow{\$} \mathrm{Enc}(\pi_s^P.cxn_{\mathtt{W}}^i.CK, \ell, \mathtt{ad}, m, \pi_s^P.cxn_{\mathtt{W}}^i.substate)$*, where* Enc *is specified by the scheme* Π *defined in* $\pi_s^P.cxn_{\mathtt{W}}^i.suite$*.*
  - *If* $(\mathtt{ad}, m) \in \mathcal{T}$*, set* $\pi_s^P.cxn_{\mathtt{W}}^i.status \leftarrow \mathtt{terminated}$*.*

  Return $(c, \pi_s^P.cxn_{\mathtt{W}}^{i\,\prime})$*.*
- $\mathrm{ChannelRcv}((\mathtt{ad}, c), \pi_s^P.cxn_{\mathtt{R}}^i)$*:*

  If $\pi_s^P.cxn_{\mathtt{R}}^i.status \neq \mathtt{active}$*, return* $(\perp, \pi_s^P.cxn_{\mathtt{W}}^i)$*. Otherwise,*
  - *compute* $(m, \pi_s^P.cxn_{\mathtt{R}}^i.substate') \leftarrow \mathrm{Dec}(\pi_s^P.cxn_{\mathtt{R}}^i.CK, \mathtt{ad}, c, \pi_s^P.cxn_{\mathtt{R}}^i.substate)$*, where* Dec *is specified by the scheme* Π *in* $\pi_s^P.cxn_{\mathtt{R}}^i.suite$*.*

  If $m = \perp$*, return* $(\perp, \pi_s^P.cxn_{\mathtt{W}}^i)$*.*

  If $(\mathtt{ad}, m) \in \mathcal{T}$*, set* $\pi_s^P.cxn_{\mathtt{R}}^i.status \leftarrow \mathtt{terminated}$ *and return* $(\perp, \pi_s^P.cxn_{\mathtt{R}}^{i\,\prime})$*.*

  Otherwise, return $((\mathtt{ad}, m), \pi_s^P.cxn_{\mathtt{R}}^{i\,\prime})$*.*

Naturally, stLHAE security of a channel protocol construction ought to be reducible to the security of the stLHAE scheme(s) $\{\Pi\}$ underlying the construction. Yet there is an additional consideration. While each Π uses connection keys derived via the KDF, all KDFs use the same shared master session key SK. Consequently, we require *agility* for $\{\mathrm{KDF}\}$. Agility for the entire set $\{\mathrm{KDF}\}$ implies that the individual primitives can share SK securely [1].

Theorem 1. *Let* CHNL$[\{(\Pi, \mathrm{KDF})\}]$ *be a keyed two-party controllable channel protocol constructed from* a_1 *stLHAE scheme(s)* $\{\Pi\}$ *and* a_2 *key derivation function(s)* $\{\mathrm{KDF}\}$*, such that* $\{\mathrm{KDF}\}$ *is a compatible, finite set that is agile with respect to pseudo-randomness. Let* $\mathcal{A}$ *be an adversarial algorithm against the* $\mathsf{Exp}^{\mathsf{stlhae}}_{\mathrm{CHNL}[\{(\Pi,\mathrm{KDF})\}]}$ *experiment in Fig. 2. Let* p *be the number of principals and* n *be the maximum number of sessions at a principal. Then we can construct a* $\{\mathrm{KDF}\}$*-restricted adversarial algorithm* $\mathcal{B}_0$ *against the pseudo-randomness agility of* $\{\mathrm{KDF}\}$*,* $\mathsf{Exp}^{\mathtt{agile-prf}}_{\mathrm{KDF.F}}$*, and adversarial algorithms* $\mathcal{B}_{i,j}$ *against* $\mathsf{Exp}^{\mathsf{stlhae}}_{\Pi_i}$*, and such that*

$$\mathbf{Adv}^{\mathsf{stlhae}}_{\mathrm{CHNL}[\{(\Pi,\mathrm{KDF})\}]}(\mathcal{A}) \leq p^2 \cdot n^2 \cdot \left(\mathbf{Adv}^{\mathtt{agile-prf}}_{\mathrm{KDF.F}}(\mathcal{B}_0) + \sum_{i=1}^{a_1}\sum_{j=1}^{a_2} \mathbf{Adv}^{\mathsf{stlhae}}_{\Pi_i}(\mathcal{B}_{i,j})\right) .$$

Due to space restrictions, the proof of Theorem 1 appears in the full version.

4 Secure Termination

Ultimately, the goal of secure termination is a guarantee to the receiver connection that no further messages are being sent. We define *closure* in the contexts of both connections and channels before presenting the secure termination adversarial advantage and experiment.

4.1 Closure Alerts and Channel Closure

Inherently, secure termination is the closure of connections, controlled by the sending and receipt of signifying messages from a subset of the message space.

Definition 9 (Connection Closure). *Let* CHNL *be a keyed two-party controllable channel protocol and* π_s^P *be a session of* CHNL*.*

- *The* i*-th write channel connection at* π_s^P *is said to be* closed *if* $\pi_s^P.cxn^i_{\mathtt{W}}.status = \mathtt{terminated}$*.*
- *The* i*-th read channel connection at* π_s^P *is said to be* closed *if* $\pi_s^P.cxn^i_{\mathtt{R}}.status = \mathtt{terminated}$*.*

If all channels at a given session are closed, the session is said to be closed*.*

Definition 10 (Channel Closure). *We say that a channel from* $\pi_s^P.cxn^i_{\mathtt{W}}$ *to* $\pi_{\bar{s}}^{P'}.cxn^i_{\mathtt{R}}$ closes using a message $m^* \in \mathcal{M}$ *if*

- $(c, \pi_s^P.cxn^{i\,\prime}_{\mathtt{W}}) \xleftarrow{\$} \mathrm{ChannelSnd}(m^*, [\ell], \pi_s^P.cxn^i_{\mathtt{W}})$*, for some* $c \in \mathcal{C}$*, where* $\pi_s^P.cxn^i_{\mathtt{W}}.status = \mathtt{active}$*, and* $\pi_s^P.cxn^{i\,\prime}_{\mathtt{W}}.status = \mathtt{terminated}$*, and*
- $(m^*, \pi_{\bar{s}}^{P'}.cxn^{i\,\prime}_{\mathtt{R}}) \leftarrow \mathrm{ChannelRcv}(p, \pi_{\bar{s}}^{P'}.cxn^i_{\mathtt{R}})$*, for some* $p \in \mathcal{P}$*, where* $\pi_{\bar{s}}^{P'}.cxn^i_{\mathtt{R}}.status = \mathtt{active}$ *and* $\pi_{\bar{s}}^{P'}.cxn^{i\,\prime}_{\mathtt{R}}.status = \mathtt{terminated}$*.*

The sending, resp. receiving, of any `terminate` *in the set of termination messages* $\mathcal{T} \subset \mathcal{M}$ *results in channel closure. Moreover, if the channel from* π_s^P *to* $\pi_{\bar{s}}^{P'}$ *closes using a* `terminate` *message, we say that* π_s^P initiated *the channel closure.*

$\underline{\mathsf{Exp}^{\mathsf{sc.t}}_{\textsc{Chnl}[\{(\Pi,\mathrm{KDF})\}],\mathcal{A}}(\lambda)}$:

1: $\mathtt{phase} \leftarrow 0$, $\mathtt{list} \leftarrow \bot$
2: $\mathcal{A}^{\mathsf{SnPairInit}(\cdot),\mathsf{ConnectionInit}(\cdot),\mathsf{Send}_{\mathsf{T}}(\cdot),\mathsf{Receive}_{\mathsf{T}}(\cdot)}()$
3: **return** $\mathtt{phase}$

$\underline{\mathsf{SnPairInit}((P,s),(P',\overline{s}))}$:

1: $SK \xleftarrow{\$} \mathrm{SessionKeyGen}(1^\kappa, P, P', s, \overline{s})$
2: **return** $\bot$

$\underline{\mathsf{ConnectionInit}(\pi_s^P, \pi_{\overline{s}}^{P'}, \Pi^*, \mathrm{KDF}^*, i)}$:

1: **if** $(\Pi^*, \mathrm{KDF}^*) \notin \{(\Pi, \mathrm{KDF})\}$ **then**
2: **return** $\bot$
3: **if** $(\pi_s^P.cxn_{\mathtt{W}}^i \in \mathtt{list}) \vee (\pi_s^P.cxn_{\mathtt{R}}^i \in \mathtt{list})$ **then**
4: **return** $(\bot, \bot)$
5: $(\pi_s^P.cxn_{\mathtt{W}}^i, \pi_s^P.cxn_{\mathtt{R}}^i) \xleftarrow{\$} \mathrm{ConnectionInit}(SK, \pi_s^P, \pi_{\overline{s}}^{P'}, \Pi^*, \mathrm{KDF}^*, i)$
6: $\mathtt{list} \leftarrow \mathtt{list} | \pi_s^P.cxn_{\mathtt{W}}^i | \pi_s^P.cxn_{\mathtt{R}}^i$
7: **return** $(\pi_s^P.cxn_{\mathtt{W}}^i, \pi_s^P.cxn_{\mathtt{R}}^i)$

$\underline{\mathsf{Send}_{\mathsf{T}}(m, [\ell], \pi_s^P.cxn_{\mathtt{W}}^i)}$:

1: $(c, \pi_s^P.cxn_{\mathtt{W}}^i) \leftarrow \mathrm{ChannelSnd}(m, [\ell], \pi_s^P.cxn_{\mathtt{W}}^i)$
2: **return** $(c, \pi_s^P.cxn_{\mathtt{W}}^i.status)$

$\underline{\mathsf{Receive}_{\mathsf{T}}(p, \pi_s^P.cxn_{\mathtt{R}}^i)}$:

1: $(m, \pi_s^P.cxn_{\mathtt{R}}^i) \leftarrow \mathrm{ChannelRcv}(p, \pi_s^P.cxn_{\mathtt{R}}^i)$
2: **if** $(m \in \mathcal{T}) \wedge (\pi_s^P.cxn_{\mathtt{R}}^i.status = \mathtt{terminated}) \wedge \big((\exists \pi_{\overline{s}}^{P'}.cxn_{\mathtt{W}}^i : \pi_{\overline{s}}^{P'}.cxn_{\mathtt{W}}^i \text{ has a channel to } \pi_s^P.cxn_{\mathtt{R}}^i) \implies (\pi_{\overline{s}}^{P'}.cxn_{\mathtt{W}}^i.status \neq \mathtt{terminated})\big)$ **then**
3: $\mathtt{phase} \leftarrow 1$
4: **return** $\pi_s^P.cxn_{\mathtt{R}}^i.status$

Fig. 3. Secure termination experiment for a protocol $\textsc{Chnl}[\{(\Pi, \mathrm{KDF})\}] = (\mathrm{SessionGen}, \mathrm{ConnectionInit}, \mathrm{ChannelSnd}, \mathrm{ChannelRcv})$ with message space $\mathcal{M}$, $\mathcal{T} \subset \mathcal{M}$, and adversary $\mathcal{A}$, where $\mathsf{Send}_{\mathsf{T}}$ and $\mathsf{Receive}_{\mathsf{T}}$ are constructed from (Π, KDF) as defined by $\pi_s^P.cxn_{\mathtt{W}}^i.suite$ and $\pi_s^P.cxn_{\mathtt{R}}^i.suite$.

Note that the sending, resp. receiving, of a `terminate` message $m \in \mathcal{T}$ results in *connection closure*; however, *channel closure* demands a causal relationship between both end connections based on m.

Remark 4. While a session may consist of multiple channel connections at any given time, it may also consist of no connections (this may occur when no channel parameters have been negotiated, or all channels have been *closed*). The number of channel connections may change during a session's lifespan, namely by means of session *closure* and *resumption*. Resumption is realized via ConnectionInit.

4.2 Secure Termination Experiment

Definition 11 (Secure Termination Experiment). *Let* Chnl *be a keyed two-party controllable channel protocol with message space $\mathcal{M}$, and let $\mathcal{A}$ be an adversary algorithm. Let* `terminate` *be an element of $\mathcal{T} \subset \mathcal{M}$, where $\mathcal{T}$ is the set of all termination messages. With the secure termination experiment for* Chnl *given by* $\mathsf{Exp}^{\mathsf{sc.t}}_{\textsc{Chnl}}$ *in Figure 3, define*

$$\mathbf{Adv}^{\mathsf{sc.t}}_{\textsc{Chnl}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathsf{sc.t}}_{\textsc{Chnl},\mathcal{A}}(\lambda) = 1\right] \quad .$$

The existence of a signifying `terminate` message (i.e. $\mathcal{T} \neq \emptyset$) is insufficient for claiming that a protocol *always* securely terminates connections. Only connection closure with such a `terminate` message yields secure termination of the connection. Intrinsically, secure termination is a property of the read side of a channel, which is achieved upon receipt of `terminate`.

Remark 5. Secure termination is a per-connection goal, achieved on message receipt, and handles adversarial intervention in channel closure – what is commonly referred to as a *truncation attack*. Truncation, malicious or otherwise, can happen via various means; for example, closure of the underlying TCP connection as a means of closing a TLS channel. The adversary wins the secure termination security game only if it is able to make an honest party accept that the connection has been correctly terminated when it has not been. Thus malicious closure of the TCP connection is not a valid attack.

Remark 6. Some protocols have "terminate"-looking messages that may not be authenticated at all or authenticated properly, and our model aids in understanding and comparing the security of such protocols against truncation attacks. For example, the `DeauthenticationRequest` of 802.11 *may* meet termination message requirements, but that is dependent on statefulness and whether or not the requests are protected as Robust Management Frames (RMF). A session that does not negotiate to send such requests as RMFs is susceptible to truncation attacks. Similarly, the security implications of 802.11's `DisassociationRequest` should be questioned. Many protocols have specified messages which appear to indicate termination; however, exactly what cryptographic guarantees are provided on receipt has not been well understood, in the absence of a secure termination model. Hence the success of truncation attacks.

Definition 12 (Terminating a Protocol). *Let* CHNL *be a two-party controllable channel protocol.*

Let π_s^P *and* $\pi_{\overline{s}}^{P'}$ *be any two sessions of* CHNL*, and let* π_s *have a channel to* $\pi_{\overline{s}}$ *according to Def. 5. If the channel from* π_s^P *to* $\pi_{\overline{s}}^{P'}$ *closes using a* `terminate` *message* `terminate`*, for* `terminate` $\in \mathcal{T}$*, according to Def. 10, and* $\mathbf{Adv}^{\mathsf{sc.t}}_{\text{CHNL}}(\mathcal{A})$ *is a negligible function in* λ *for all PPT adversaries* $\mathcal{A}$*, then* CHNL *is said to* achieve secure termination on the channel from π_s to $\pi_{\overline{s}}$ *with* `terminate`*.*

Naturally, the fact that the read connection of the closure-initiator's session may not necessarily be required to close when the write connection sends a `terminate` message gives rise to the concepts of *fatal* and *graceful secure termination*. When a session π_s^P initiates closure with `terminate` $\in \mathcal{T}$ such that only the write connection closes, graceful closure *can* be achieved by waiting for a corresponding `terminate` message to be sent to the read connection of π_s^P before the read connection closes. Thus both sessions sharing the channels *may* achieve secure termination. Comparatively, if π_s^P initiates closure with a fatal `terminate` message, it *cannot* achieve secure termination – only the receiving session may achieve it.

Definition 13 (Fatal and Graceful Secure Termination). *Let* π_s^P *and* $\pi_{\overline{s}}^{P'}$ *be sessions of a two-party controllable channel protocol* CHNL *such that* π_s^P *has an channel to* $\pi_{\overline{s}}^{P'}$ *according to Def. 5. Let* `terminate` $\in \mathcal{T}$*.*

– *If, upon running* $(c, \pi_s^P.cxn_{\mathtt{W}}^{i\ \prime}) \xleftarrow{\$} \text{ChannelSnd}(\mathtt{terminate}, \pi_s^P.cxn_{\mathtt{W}}^i)$*, both the write and read connections at* π_s^P *are closed according to Def. 9, then* `terminate` *is said to be a* fatal termination message.
– *If, upon running* $(c, \pi_s^P.cxn_{\mathtt{W}}^{i\ \prime}) \xleftarrow{\$} \text{ChannelSnd}(\mathtt{terminate}, \pi_s^P.cxn_{\mathtt{W}}^i)$*, only the write connection at* π_s *is closed according to Def. 9, then* `terminate` *is said to be a* graceful termination message.

4.3 Reduction to stLHAE Security

Secure termination depends upon the relay of messages with unaltered content and therefore its security is reducible to that of the authentication guarantees of the channel.

Theorem 2. *Let* $\textsc{Chnl}[\{(\Pi, \text{KDF})\}]$ *be a keyed two-party controllable channel protocol constructed from stLHAE scheme(s)* $\{\Pi\}$*, with a message space* $\mathcal{M}$ *and* $\mathcal{T} \subset \mathcal{M}$*, and let* $\mathcal{A}$ *be an adversarial algorithm against the* $\mathsf{Exp}^{\mathsf{sc.t}}_{\textsc{Chnl}[\{(\Pi,\text{KDF})\}]}$ *experiment. Then we can construct an adversarial algorithm* $\mathcal{B}$ *against* $\mathsf{Exp}^{\mathsf{stlhae}}_{\textsc{Chnl}[\{(\Pi,\text{KDF})\}]}$ *such that*

$$\mathbf{Adv}^{\mathsf{sc.t}}_{\textsc{Chnl}[\{(\Pi,\text{KDF})\}]}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{stlhae}}_{\textsc{Chnl}[\{(\Pi,\text{KDF})\}]}(\mathcal{B}) \ .$$

Due to space restrictions, the proof of Theorem 2 appears in the full version.

5 Secure Channels and Termination in TLS 1.2

TLS is one of the most important security protocols in the world today and serves as the backbone of internet security. Due to the lack of key indistinguishability in TLS 1.2, many works analyzing it rely on the ACCE model [20]. We show that the *post-accept* phase channel ACCE security model can be viewed as a highly restricted case of channel protocol stateful length-hiding AE (stLHAE) security, and provide positive secure termination results for TLS 1.2 under this restricted case. Note that the use of the *post-accept* ACCE model, together with the coinciding channel restrictions, is not inherent; we demonstrate the correlation to bypass a reanalysis of TLS and directly consider secure termination. Analyzing TLS in our channel protocol model would allow for consideration of parallel sessions and connections, as well as resumption, key derivation functions, etc.

5.1 Comparing Channel Protocols and ACCE

ACCE employs a *pre-accept* phase and a *post-accept* phase. While the former handles all protocol interactions before a session key has been accepted, the latter is correlated to channel security under the agreed-upon session key. Breaking ACCE security is described in terms of either getting a session oracle to accept maliciously, or correctly answering a stLHAE encryption challenge.

Since Chnl session keys need not be sampled uniformly at random, it is possible to adapt the Chnl stLHAE construction (Def. 8) and allow for SK to be derived by other means (e.g. as the output of a handshake in the ACCE *pre-accept* phase). Thus we can link the Chnl stLHAE security experiment to

the ACCE *post-accept* phase experiment (see [20]). The ACCE model uses the session key directly for the stLHAE primitive, with key derivation consisting of splitting the session key into two in order to obtain write/read keys. Since only one session key exists and is used directly as the connection keys (e.g. on $cxn_{\mathtt{W}}$, $cxn_{\mathtt{R}}$), only one connection pair is allowed per session. Another session key would be required for further connections if key separation is to be achieved. This assumption is inherent to the ACCE model TLS analyses to date, where SK is assumed to be a concatenation of the write and read keys (e.g. [23]).

Theorem 3. *Let* $\textsc{Chnl}[(\Pi, [])]$ *be a keyed two-party controllable channel protocol constructed from a single stLHAE scheme* Π*, using the identity* KDF []*. Let* p *be the number of principals,* n *be the maximum number of sessions at a principal, and let* $\mathcal{A}$ *be an adversarial algorithm against the* $\mathsf{Exp}^{\mathsf{stlhae}}_{\textsc{Chnl}[(\Pi,[])]}()$ *experiment in Fig. 2. Then we can construct an adversarial algorithm* $\mathcal{B}$ *against* $\mathsf{Exp}^{ACCE}_{\Pi}()$ *such that*

$$\mathbf{Adv}^{\mathsf{stlhae}}_{\textsc{Chnl}[(\Pi,[])]}(\mathcal{A}) \leq p^2 \cdot n^2 \cdot \mathbf{Adv}^{ACCE}_{\Pi}(\mathcal{B}) \ .$$

Due to space restrictions, the proof of Theorem 3 appears in the full version.

5.2 Secure Termination in TLS

According to specification, the Alert Protocol in TLS falls into three categories: fatal alerts, warning alerts, and `close_notify` alerts. Unlike fatal alerts, which upon sending/receipt close both write and read connections at a session, `close_notify` alerts do not necessarily close the initiator's receive connection immediately, but *may* wait until receipt of the reciprocal `close_notify` alert. Fig. 4 and 5 illustrate fatal and `close_notify` alert behavior. According to the TLS 1.2 standard, the determination of whether or not the initiator's read connection should be closed when a `close_notify` is sent is left to the usage profile [11, p. 28]. However, upon receipt of a `close_notify` alert, the responder *mus*t close its read connection and *must* send a corresponding `close_notify` alert on its send connection, before closing it also. Receipt of the reciprocal alert results in the closure of the original initiator's read connection.

Ostensibly, TLS fatal alerts are *fatal termination messages* per Def. 13 and the TLS 1.2 specification [11]. Comparatively, `close_notify` alerts are either *fatal* or *graceful termination messages*, depending on the implementation. Per specification, "It is not required for the initiator of the close to wait for the responding `close_notify` alert before closing the read side of the connection."

Remark 7. It is crucial to note that Alert Protocol messages are in fact, within the message space of the TLS protocol according to Def. 5 and Def. 8, despite the Alert Protocol running on top of the Record Layer Protocol. This is due to the fact that TLS 1.2 encrypts and sequences these messages in the same manner as application messages [11, section 7.2].

We conclude with the following theorem on secure termination in TLS, under the restricted (ACCE-induced) channel environment of prior TLS analyses.

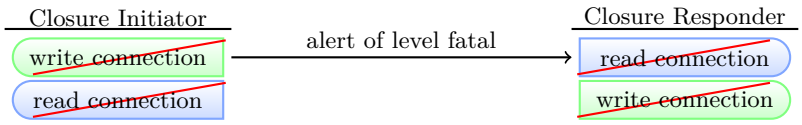

Fig. 4. TLS fatal alert behavior, where *initiator* is the session initiating a channel closure. Both the read and write *cxn* of the initiator close immediately upon sending.

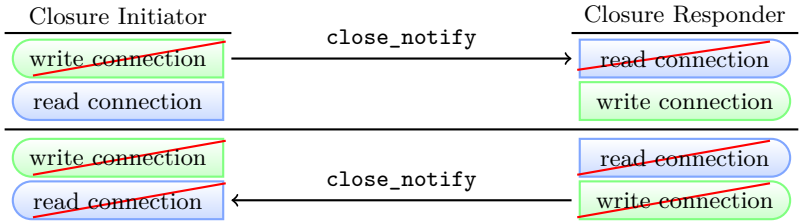

Fig. 5. TLS `close_notify` behavior. The write *cxn* of the initiator closes upon sending of `close_notify`. Upon receipt, the read *cxn* at the responder closes. Subsequently, the responder sends a corresponding `close_notify` alert and closes its write *cxn*.

Theorem 4. *For TLS 1.2, TLS-RSA, TLS-CCA, TLS-DH, and TLS-DHE achieve secure termination under any fatal alert or* `close_notify` *alert, where each TLS session consists of a single connection pair with connection keys derived from the session key via the identity* KDF*.*

Theorem 4 follows by combining previous ACCE analyses of TLS 1.2 [29, 20, 23, 7], Theorem 3, and Theorem 2, and applying the observation above that *fatal* and `close_notify` alerts satisfy the definition of `terminate` messages within $\mathcal{T} \subset \mathcal{M}$, where $\mathcal{M}$ is the protocol message space.

From previous analyses of TLS 1.2 in the ACCE model, Thm. 4 inherits the restriction that sessions consist of a single connection pair. However this is not intrinsic to the design of TLS and it may be analyzed in the keyed two-party controllable channel model for better consideration of the full protocol.

References

1. Acar, T., Belenkiy, M., Bellare, M., Cash, D.: Cryptographic agility and its relation to circular encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 403–422. Springer, Heidelberg (May 2010)
2. Badertscher, C., Matt, C., Maurer, U., Rogaway, P., Tackmann, B.: Augmented secure channels and the goal of the TLS 1.3 record layer. In: Au, M.H., Miyaji, A. (eds.) ProvSec 2015. LNCS, vol. 9451, pp. 85–104. Springer, Heidelberg (Nov 2015)
3. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Journal of Cryptology 21(4), 469–491 (Oct 2008)
4. Bergsma, F., Dowling, B., Kohlar, F., Schwenk, J., Stebila, D.: Multi-Ciphersuite Security of the Secure Shell (SSH) Protocol. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 369–381. CCS '14, ACM (2014), `http://doi.acm.org/10.1145/2660267.2660286`

5. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy. pp. 98–113. IEEE Computer Society Press (May 2014)
6. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zanella Béguelin, S.: Proving the TLS handshake secure (as it is). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 235–255. Springer, Heidelberg (Aug 2014)
7. Boyd, C., Hale, B., Mjølsnes, S.F., Stebila, D.: From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 55–71. Springer, Heidelberg (Feb / Mar 2016)
8. Brier, E., Peyrin, T.: A forward-secure symmetric-key derivation protocol - how to improve classical DUKPT. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 250–267. Springer, Heidelberg (Dec 2010)
9. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (May 2001)
10. Cohen, R., Coretti, S., Garay, J., Zikas, V.: Probabilistic Termination and Composability of Cryptographic Protocols (2016), `http://eprint.iacr.org/`
11. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2 (2008), `https://tools.ietf.org/html/rfc5426`, RFC 5426
12. Fischlin, M., Günther, F., Marson, G.A., Paterson, K.G.: Data is a stream: Security of stream-based channels. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 545–564. Springer, Heidelberg (Aug 2015)
13. Giesen, F., Kohlar, F., Stebila, D.: On the security of TLS renegotiation. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 387–398. ACM Press (Nov 2013)
14. Günther, F., Mazaheri, S.: A Formal Treatment of Multi-key Channels. In: Proceedings of the 37th International Cryptology Conference. CRYPTO'17, Springer (2017)
15. Haber, S., Pinkas, B.: Securely combining public-key cryptosystems. In: ACM CCS 01. pp. 215–224. ACM Press (Nov 2001)
16. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 15–44. Springer, Heidelberg (Apr 2015)
17. Hoang, V.T., Reyhanitabar, R., Rogaway, P., Vizár, D.: Online authenticated-encryption and its nonce-reuse misuse-resistance. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 493–517. Springer, Heidelberg (Aug 2015)
18. Hoepman, J.H.: The ephemeral pairing problem. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 212–226. Springer, Heidelberg (Feb 2004)
19. IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (2012), `http://dx.doi.org/10.1109/IEEESTD.2012.6178212`
20. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012)
21. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (Apr 2001)

22. Krawczyk, H.: Cryptographic extraction and key derivation: The HKDF scheme. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 631–648. Springer, Heidelberg (Aug 2010)
23. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer, Heidelberg (Aug 2013)
24. Marson, G., Poettering, B.: Security Notions for Bidirectional Channels. In: IACR Transactions on Symmetric Cryptology. vol. 2017, pp. 405–426. `http://ojs.ub.rub.de/index.php/ToSC/article/view/602`
25. Maurer, U., Tackmann, B.: On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 10. pp. 505–515. ACM Press (Oct 2010)
26. Maurer, U.M.: Perfect cryptographic security from partially independent channels. In: 23rd ACM STOC. pp. 561–571. ACM Press (May 1991)
27. Microsoft-Inria Joint Centre: miTLS: A verified reference TLS implementation (2012), `https://www.mitls.org/pages/attacks`
28. Namprempre, C.: Secure channels based on authenticated encryption schemes: A simple characterization. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 515–532. Springer, Heidelberg (Dec 2002)
29. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the TLS record protocol. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 372–389. Springer, Heidelberg (Dec 2011)
30. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3: draft-ietf-tls-tls13-20 (April 2017), `https://tools.ietf.org/pdf/draft-ietf-tls-tls13-20.pdf`, expires October 30, 2017
31. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security (2006), `https://tools.ietf.org/html/rfc4347`, RFC 4347
32. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 02. pp. 98–107. ACM Press (Nov 2002)
33. Rogaway, P., Shrimpton, T.: Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. Cryptology ePrint Archive, Report 2006/221 (2006), `http://eprint.iacr.org/2006/221`
34. Shrimpton, T.: A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272 (2004), `http://eprint.iacr.org/2004/272`
35. Smyth, B., Pironti, A.: Truncating TLS connections to violate beliefs in web applications. In: 7th USENIX Workshop on Offensive Technologies, WOOT '13 (2013)

A stLHAE Syntax and Security

Definition 14 (Stateful Length-Hiding AEAD). *A* stateful length-hiding AEAD scheme Π *for a message space* $\mathcal{M}$*, an associated data space* $\mathcal{AD}$*, a key space* $\mathcal{K}$*, and a ciphertext space* $\mathcal{C}$*, is a tuple of algorithms:*

- $\text{Kgn}() \xrightarrow{\$} k$*: A probabilistic key generation algorithm that outputs a key* k*.*
- $\text{Enc}(k, \ell, \mathtt{ad}, m, st_{\text{S}}) \xrightarrow{\$} (c, st'_{\text{S}})$*: A probabilistic encryption algorithm that takes as input a key* $k \in \mathcal{K}$*, a length* $\ell \in \mathbb{Z} \cup \{\perp\}$*, associated data* $\mathtt{ad} \in \mathcal{AD}$*, a message* $m \in \mathcal{M}$*, and an write state* st_{S}*, and outputs a ciphertext* $c \in \mathcal{C}$ *or an error symbol* $\perp$*, and updated state* st'_{S}*.*

– $\mathrm{Dec}(k, \mathtt{ad}, c, st_{\mathrm{R}}) \to (m, st'_{\mathrm{R}})$: *A deterministic decryption algorithm that takes as input a key $k \in \mathcal{K}$, associated data $\mathtt{ad} \in \mathcal{AD}$, a ciphertext c, and a read state st_{R}, and outputs a message $m \in \mathcal{M}$ or an error symbol $\perp$, and an updated state st'_{R}.*

If $\ell \neq \perp$, then we say that Π is length-hiding.

Correctness is defined in the obvious way, based on scheme correctness from Def. 1.

Definition 15 (Stateful Length-Hiding AEAD Security). *Let Π be a stateful length-hiding AEAD scheme and let $\mathcal{A}$ be an PPT adversarial algorithm. The stateful length-hiding AEAD experiment for Π with bit b is given by $\mathsf{Exp}^{\mathsf{stlhae}}_{\Pi}(\mathcal{A})$ in Fig. 6. We define*

$$\mathbf{Adv}^{\mathsf{stlhae}}_{\Pi}(\mathcal{A}) = 2\Pr\left[\mathsf{Exp}^{\mathsf{stlhae}}_{\Pi}(\mathcal{A})\right] - 1 \ .$$

Note that the state variables st_{S}, st_{R} in Fig. 6 are considered substate variables in terms of the channel environment (e.g. $\pi^P_s.cxn^i_{\mathtt{W}}.substate$, $\pi^P_s.cxn^i_{\mathtt{R}}.substate$). This is due to the increased state considerations of the environment.

$\underline{\mathsf{Exp}^{\mathsf{stlhae}}_{\Pi}(\mathcal{A})}$:
1: $k \xleftarrow{\$} \mathrm{Kgn}()$
2: $b \xleftarrow{\$} \{0,1\}$
3: $st_{\mathrm{S}} \leftarrow \perp, st_{\mathrm{R}} \leftarrow \perp$
4: $u \leftarrow 0,\ v \leftarrow 0$
5: $\mathtt{phase} \leftarrow 0$
6: $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Encrypt}(\cdot),\mathsf{Decrypt}(\cdot)}()$
7: **return** $(b' = b)$

$\underline{\mathsf{Enc}(\ell, \mathtt{ad}, (m_0, m_1))}$:
1: $u \leftarrow u + 1$
2: $(sent.c^{(0)}, st^{(0)}_{\mathrm{S}}) \leftarrow \mathrm{Enc}(k, \ell, \mathtt{ad}, m_0, st_{\mathrm{S}})$
3: $(sent.c^{(1)}, st^{(1)}_{\mathrm{S}}) \leftarrow \mathrm{Enc}(k, \ell, \mathtt{ad}, m_1, st_{\mathrm{S}})$
4: **if** $sent.c^{(0)} = \perp$ or $sent.c^{(1)} = \perp$ **then**
5: **return** $\perp$
6: $(sent.ad_u, sent.c_u, st_{\mathrm{S}}) := (\mathtt{ad}, sent.c^{(b)}, st^{(b)}_{\mathrm{S}})$
7: **return** $sent.c_u$

$\underline{\mathsf{Dec}(\mathtt{ad}, c)}$:
1: **if** $b = 0$ **then**
2: **return** $\perp$
3: $v \leftarrow v + 1$
4: $(m, st_{\mathrm{R}}) \leftarrow \mathrm{Dec}(k, \mathtt{ad}, c, st_{\mathrm{R}})$
5: **if** $(u < v) \vee (c \neq sent.c_v) \vee (\mathtt{ad} \neq sent.ad_v)$ **then**
6: $\mathtt{phase} \leftarrow 1$
7: **if** $\mathtt{phase} = 1$ **then**
8: **return** m
9: **return** $\perp$

Fig. 6. Stateful length-hiding AEAD experiment stlhae for stateful length-hiding AEAD scheme $\Pi = (\mathrm{Kgn}, \mathrm{Enc}, \mathrm{Dec})$ and adversary $\mathcal{A}$.