Augmented Secure Channels and the Goal of the TLS 1.3 Record Layer

Christian Badertscher[1], Christian Matt[1], Ueli Maurer[1],
Phillip Rogaway[2], and Björn Tackmann[3]

[1]Department of Computer Science, ETH Zurich, Switzerland
[2]Department of Computer Science, University of California, Davis, USA
[3]Department of Computer Science & Engineering,
University of California, San Diego, USA

Abstract

Motivated by the wide adoption of authenticated encryption and TLS, we suggest a basic channel abstraction, an *augmented secure channel* (ASC), that allows a sender to send a receiver messages consisting of two parts, where one is privacy-protected and both are authenticity-protected. Working in the tradition of constructive cryptography, we formalize this idea and provide a construction of this kind of channel using the lower-level tool authenticated-encryption.

We look at recent proposals on TLS 1.3 and suggest that the criterion by which their security can be judged is quite simple: do they construct an ASC? Due to this precisely defined goal, we are able to give a natural construction that comes with a rigorous security proof and directly leads to a proposal on TLS 1.3 that, in addition to being provably secure, is more efficient than existing ones.

1 Introduction

This paper defines and investigates a new abstraction of a secure channel. We call it an *augmented secure channel*, or ASC. Like most types of channels, an ASC lets a sender Alice send messages to a receiver Bob. But unlike more conventional types of channels, each message has designated private and non-private parts. An active adversary Eve occupies the system, but is limited to seeing the length of the private portion and the contents of the non-private portion of each message—and to entirely shutting down the channel. In particular, the adversary cannot inject messages or induce out-of-order message delivery. Additionally, the non-private portion can contain an implicit part, already known to the receiver, that is not transmitted but still authenticated, e.g., to bind the message to a given context.

The service an ASC provides is motivated by the ascendancy of both TLS and authenticated encryption. We take the rise of these tools, and what they deliver, as an indication that customary conceptualizations of secure channels may not have been rich enough to deliver the service that protocol designers routinely need.

Authenticated encryption. While ASCs are closely related to schemes for authenticated encryption (AE) and authenticated encryption with associated data (AEAD), an ASC and an AE/AEAD-scheme are very different things. An ASC is a reasonably high-level object: an abstract *resource* that parties can employ, getting compositional guarantees when they do. Our formulation of ASCs will be in the tradition of *constructive cryptography* [16, 17]. In contrast, an AEAD-scheme is a comparatively low-level primitive: it is a tuple of algorithms that is "good" in some particular, complexity-theoretic sense.

The AEAD notion emerged over a sequence of works [2, 3, 12, 13, 22, 23, 24] having two distinct purposes: to minimize the misuse of symmetric encryption primitives and to gain efficiency advantages over generic composition schemes (i.e., traditional ways to meld privacy-only encryption schemes and message-authentication codes). But in moving from conventional encryption to AEAD, the basic conception of what a symmetric encryption *is* was thoroughly revamped: authenticity became an intrinsic part of the goal; so too did the allowance of (non-private) associated data A; while probabilism, formerly seen as indispensable, was surfaced and subsumed by a nonce N. Roughly said, an AEAD-scheme would nowadays be defined as a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where a computationally-reasonable adversary $\mathcal{A}$ has poor advantage at distinguishing encryption and decryption oracles $(\mathcal{E}_K(N, A, M), \mathcal{D}_K(N, A, C))$ from a pair of oracles $(\$(N, A, M), \bot(N, A, C))$, where K is generated by $\mathcal{K}$, the $\$(N, A, M)$ oracle returns an appropriate number of random bits, the $\bot(N, A, C)$ oracle always returns $\bot$, the adversary repeats no nonce N in queries to its first oracle, and queries that would result in trivial wins are disallowed.

The new conceptualization for symmetric encryption gained surprising rapid acceptance. The IEEE, IETF, ISO, and NIST all stepped in to standardize AEAD-schemes (e.g., in NIST SP 800-38C and SP 800-38D, IEEE 802.11i, ISO 19772, and IETF RFC 3610, 5116, 5288, 5297, and 7253). Methods that had been previously embedded in widely-deployed systems and standards, (e.g., Kerberos, SSH, and SSL) were recognized as attempts—sometimes rather clumsy ones—to achieve AE/AEAD. Revisions to widely-used protocols started to deploy the ready-built solutions to AEAD rather than the *ad hoc* and error-prone mechanisms that had provided no real abstraction boundaries other that of block ciphers, hash functions, or MACs.

Understanding the goal of TLS. A lot of previous work has focused on proving the security of TLS (mainly for versions prior to 1.3) [8, 9, 11, 14, 15, 19, 20, 25]. Motivated by the adoption of AEAD in TLS 1.3, we give a novel interpretation for the goal of the TLS record layer: constructing a specific instantiation of an ASC. Indeed, messages in the TLS record protocol consist of private and non-private parts, which are both authenticated.

We show how a generic ASC construction directly leads to this specific instantiation of an ASC. We thereby obtain a provably secure TLS record protocol. This protocol improves existing proposals on TLS 1.3 by reducing the size of transmitted records and the number of elements in the authenticated data.

The gap between definitions and their use. Classical cryptographic definitions, including the AEAD definition reviewed above, do not capture in which contexts a

scheme satisfying them can securely be used. They consider a specific attack model and give certain capabilities to an adversary that tries to win some game, but it is not *a priori* clear which capabilities an adversary has in a particular application, or even what her final goal is. To illustrate our point, consider the standard notions for encryption schemes, IND-CPA and IND-CCA. While IND-CCA is stronger, it is not obvious in which applications an IND-CCA encryption scheme is needed and where IND-CPA would suffice. These considerations are highly security-relevant. For complex protocols like TLS or IPSec, one has to make sure that any overall attack can be translated to an attack against the CPA or CCA game or another hardness assumption; only then the protocol is sound. But such analyses are complex and cannot be reused for the analysis of other protocols or attack models.

To solve this problem, we divide a complex protocol into several less complex *construction steps*. Each step specifies precisely what is assumed and what is achieved. Following the tradition of constructive cryptography[1] (CC) [16, 17], we model guarantees and expectations as *resources* that provide a specified service to each party. Every party possesses an *interface* to the resources via which it can request that service. We consider resources with three interfaces, labeled A (for Alice), B (for Bob) and E (for Eve). The construction notion of CC provides the following compositional guarantee: a constructed resource (e.g., an ASC) can be used in any other construction as an assumed resource. We obtain modularity in the sense that the overall security follows automatically from individual security proofs.

The approach has already been applied successfully in many other contexts. For example, the results in [6, 7] shed new light on the definitions of public-key encryption schemes and even led to a new security definition.

2 Preliminaries

Notation. We describe our systems with pseudocode using the following conventions: We write $x \leftarrow y$ for assigning the value y to the variable x. For a distribution $\mathcal{D}$ over some set, $x \twoheadleftarrow \mathcal{D}$ denotes sampling x according to $\mathcal{D}$. For a finite set X, $x \twoheadleftarrow X$ denotes assigning to x a uniformly random value in X. Typically queries to systems consist of a suggestive keyword and a list of arguments (e.g., (send, M) to send the message M). We ignore keywords in writing the domains of arguments, e.g., $(\mathsf{send}, M) \in \mathcal{M}$ indicates that $M \in \mathcal{M}$.

AEAD. Let Σ be an alphabet (a finite nonempty set). Typically an element of Σ is a bit ($\Sigma = \{0,1\}$) or a byte ($\Sigma = \{0,1\}^8$). For a string $x \in \Sigma^*$, $|x|$ denotes its length. We define the syntax of a scheme for authenticated encryption with associated data (AEAD) following [22].

Definition 1. An AEAD-scheme Π is a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ is a randomized algorithm that samples a key $K \in \Sigma^*$, $\mathcal{E}$ is a deterministic algorithm that maps a key $K \in \Sigma^*$, a nonce $N \in \mathcal{N}$, additional data $A \in \mathcal{A}$, and a message $M \in \mathcal{M}$ to a ciphertext $C \in \mathcal{C}$, and $\mathcal{D}$ is a deterministic algorithm that maps $(K, N, A, C) \in$

---

[1] We suspect that alternative definitional frameworks, like treating ASCs in the UC framework [5] or RSIM [1, 21], would yield closely related findings.
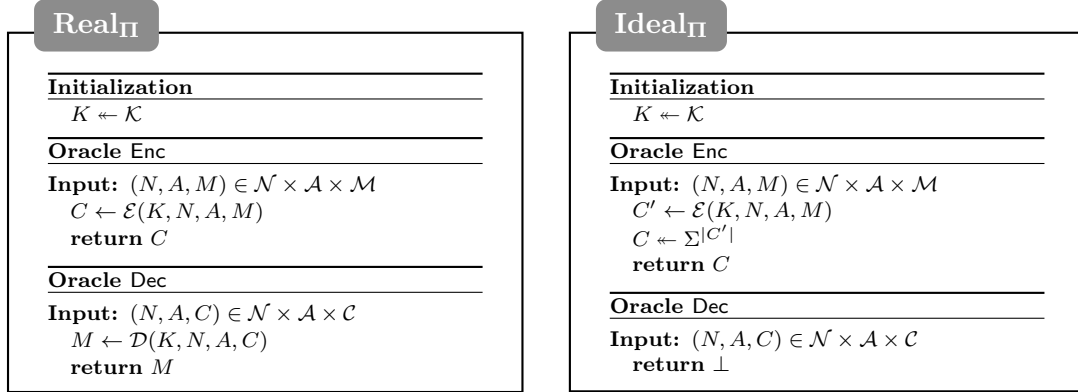
```
Realᴨ

Initialization
    K ↞ 𝒦

Oracle Enc
Input: (N, A, M) ∈ 𝒩 × 𝒜 × ℳ
    C ← ℰ(K, N, A, M)
    return C

Oracle Dec
Input: (N, A, C) ∈ 𝒩 × 𝒜 × 𝒞
    M ← 𝒟(K, N, A, C)
    return M
```

```
Idealᴨ

Initialization
    K ↞ 𝒦

Oracle Enc
Input: (N, A, M) ∈ 𝒩 × 𝒜 × ℳ
    C′ ← ℰ(K, N, A, M)
    C ↞ Σ|C′|
    return C

Oracle Dec
Input: (N, A, C) ∈ 𝒩 × 𝒜 × 𝒞
    return ⊥
```

Figure 1: Real and ideal security game for AEAD-schemes.

$\Sigma^* \times \mathcal{N} \times \mathcal{A} \times \mathcal{C}$ to $\mathcal{M} \cup \{\bot\}$. We assume the domains $\mathcal{N}, \mathcal{A}, \mathcal{M}$, and $\mathcal{C}$ are equal to Σ^* and require for all $K, N, A, M \in \Sigma^*$ that $\mathcal{D}\big(K, N, A, \mathcal{E}(K, N, A, M)\big) = M$. We further require the length of a ciphertext $|\mathcal{E}(K, N, A, M)|$ only depend on the length of the corresponding message $|M|$.

We define the security game for AEAD-schemes using the all-in-one formulation from [10]. A scheme is considered secure if all valid and efficient adversaries $\mathcal{A}$ have poor advantage according to the following definition.

Definition 2. We define the advantage of an adversary $\mathcal{A}$ as the difference in the probability that it outputs 1 in the real and ideal games defined in Figure 1:

$$\mathbf{Adv}_\Pi^{\mathrm{ae}}(\mathcal{A}) := \Pr\big[\mathcal{A}^{\mathbf{Real}_\Pi} = 1\big] - \Pr\big[\mathcal{A}^{\mathbf{Ideal}_\Pi} = 1\big].$$

An adversary is *valid* if it does not repeat Enc or Dec queries, does not ask queries $\mathsf{Enc}(N, A, M)$ and $\mathsf{Enc}(N, A', M')$ (i.e., does not repeat nonces), and does not ask a query $\mathsf{Dec}(N, A, C)$ where C was returned by a preceding query $\mathsf{Enc}(N, A, M)$.

3 Revisiting the Functionality and Modeling of Communication Channels

In constructive cryptography, communication channels are modeled as resources with three interfaces: Interface A for sender Alice, interface B for receiver Bob, and interface E for adversary Eve. Different types of such channels have been studied that differ in the capabilities of the adversary Eve [7, 16, 18].

In the following paragraphs, we present a formalization of secure and insecure channels and argue why in many applications users might need more services than provided by either. To resolve this mismatch, we introduce a new type of channel, which we call *augmented secure channel (ASC)*, that provides those missing functionalities.

3.1 Existing Formalizations

Insecure channel. The insecure channel **IC** allows messages to be input repeatedly at interface A. Each message is subsequently leaked at the E-interface. At interface E,

Resource IC

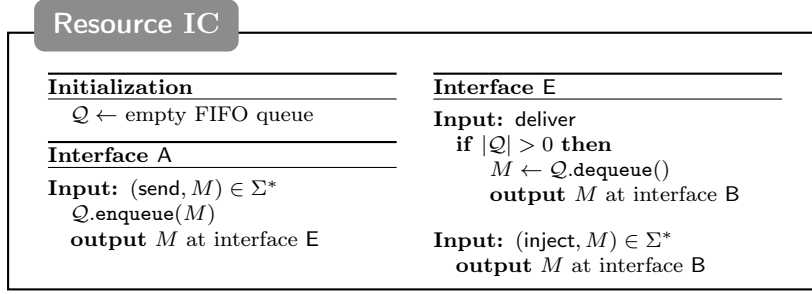| Initialization	Interface E
$\mathcal{Q} \leftarrow$ empty FIFO queue	**Input:** deliver
	if $|\mathcal{Q}| > 0$ **then**
Interface A	$M \leftarrow \mathcal{Q}.\mathsf{dequeue}()$
	output M at interface B
Input: $(\mathsf{send}, M) \in \Sigma^*$	
$\mathcal{Q}.\mathsf{enqueue}(M)$	**Input:** $(\mathsf{inject}, M) \in \Sigma^*$
output M at interface E	**output** M at interface B

Figure 2: The insecure channel resource.

arbitrary messages (including those that were previously input at interface A) can be injected such that they are delivered to B. This channel does not give any security guarantees to Alice and Bob. A formal description is provided in Figure 2.

Secure channel. The typical formalization of a secure channel follows the same basic structure as an insecure channel but where the ability of the adversary is limited to seeing the length of the transmitted messages and to deliver messages input at interface A. In particular, the adversary cannot inject new messages or induce out-of-order message delivery. A description of the secure channel can be derived from Figure 2 by omitting the inject-query and by restricting the leakage at interface E to $|M|$ on inputs (send, M) at interface A.

3.2 What Service Should a Secure Channel Provide?

In many relevant security protocols, like TLS, transmitted data packets are usually divided into a header part and a payload part. While both are required to be authentic, only the payload has to remain confidential.

We further observe that the header often contains context information since binding a message to a given context is good security-engineering practice. Moreover, parts of the context are already known to the receiver. This part does not have to be transmitted but should still be authenticated. This suggests splitting the header into two parts: an explicit part and an implicit part that describes the unknown and known parts of the header.

We conclude that there is a need for an abstract functionality that allows one to transmit a message together with the explicit part of a header such that the message remains private and the message as well as both the explicit and the implicit part of the header are authenticated.

Augmented secure channel. We now present the channel abstraction that formalizes the desired service. The augmented secure channel **ASC** is described in Figure 3: The sender can provide a triple consisting of the explicit part of a header $E \in \mathcal{H}_{\mathrm{E}}$, the implicit part of the header $I \in \mathcal{H}_{\mathrm{I}}$, and a message $M \in \mathcal{M}$. The message remains confidential and the explicit part of the header is leaked at the adversarial interface. If the receiver knows the implicit part of the header, he can recover the message using the query (fetch, I) and verify the authenticity of the message and both parts of the header.

```
┌─ Resource ASC ──────────────────────────────────────────────────────────────┐
│                                                                              │
│  Initialization                          Interface E                         │
│    S ← empty FIFO queue                 Input: deliver                        │
│    R ← empty FIFO queue                   if |S| > 0 and halt = 0 then        │
│    halt ← 0                                   (E, I, M) ← S.dequeue()         │
│  ──────────────────────────               R.enqueue((E, I, M))               │
│  Interface A                                  output (newMsg, E) at interface B│
│  Input: (send, E, I, M) ∈ H_E × H_I × M                                      │
│    S.enqueue((E, I, M))                  Input: (injectStop, E) ∈ H_E         │
│    output (E, |M|) at interface E          if halt = 0 then                   │
│  ──────────────────────────                   R.enqueue((⊥, ⊥, ⊥))           │
│  Interface B                                  output (newMsg, E) at interface B│
│  Input: (fetch, I) ∈ H_I                                                      │
│    if |R| > 0 and halt = 0 then                                              │
│        (E', I', M') ← R.dequeue()                                            │
│        if I' = I ≠ ⊥ then                                                    │
│            output M' at interface B                                          │
│        else                                                                  │
│            halt ← 1                                                          │
│            output ⊥ at interface B                                           │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```
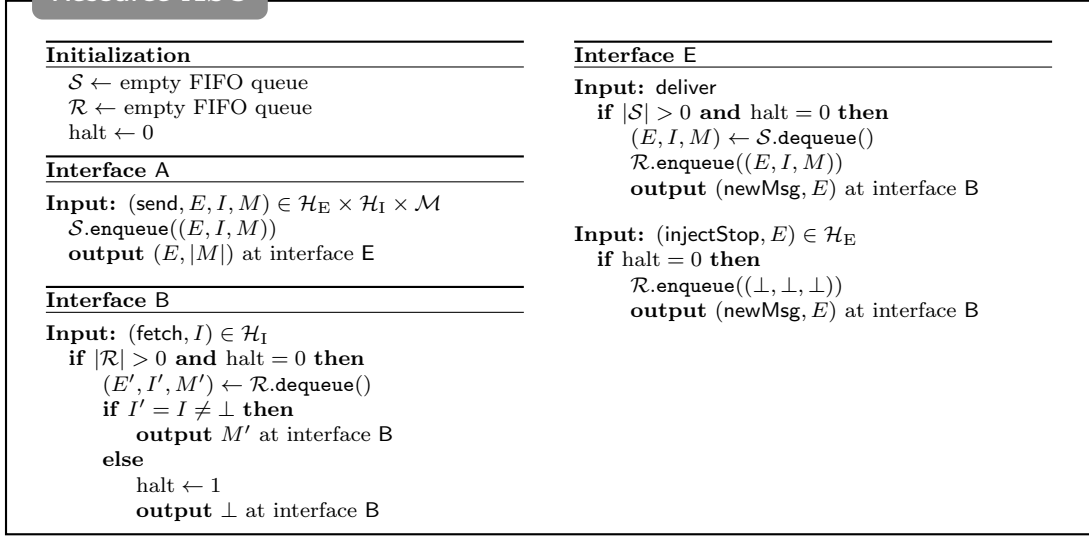
Figure 3: Description of **ASC**, an augmented secure channel.

If the verification fails, the system stops delivering messages and signals an error by outputting $\perp$. The adversary has the ability to deliver messages and to inject a special element that will terminate the channel at the receiver's side once fetched. Delivering a message notifies the receiver of the new message and provides him with the explicit part of the header.

4 Constructing an Augmented Secure Channel via Authenticated Encryption

After motivating the need for the new channel **ASC**, we now show how to construct it using an AEAD-scheme. We first introduce the assumed resources from which we construct **ASC** and describe the protocol that achieves this construction. We finally prove the security of our construction.

4.1 Assumed Resources

We construct the augmented secure channel from an insecure channel **IC** and a shared secret key. We introduce a shared key resource $\mathbf{SK}_{\mathcal{K}}$ for some key distribution $\mathcal{K}$ that initially chooses a key according to $\mathcal{K}$ and outputs this key to interfaces A and B while interface E remains inactive. See Figure 4. We denote by $[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ the resource that provides at each interface access to the corresponding interface of both the key and the channel.

4.2 Protocol

A protocol is modeled in constructive cryptography as a pair of *converters* that specify the actions of both honest parties Alice and Bob. A converter is a system with two interfaces: the *inner interface* in is connected to an interface of a resource and the
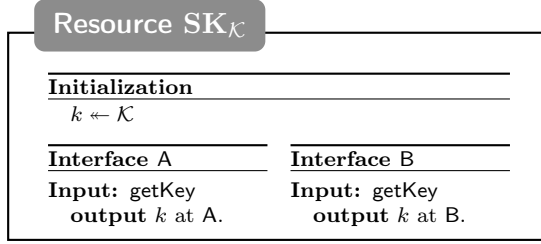
Figure 4: The shared secret key resource.

outer interface out becomes the new connection point of that resource towards the environment. Attaching a converter to an interface changes the local behavior at that interface and hence yields a new resource.[2]

For an AEAD-scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, we present the protocol $(\mathsf{enc}_\Pi, \mathsf{dec}_\Pi)$ as pseudocode in Figure 5. The converter for the sender, enc_Π, accepts inputs of the form (send, E, I, M) at its outer interface and encrypts the message M using $\mathcal{E}$. The nonce is implemented as a counter, the additional data[3] is $H = (E, I)$ and the key is provided by the key-resource $\mathbf{SK}_\mathcal{K}$. An encoding of the resulting ciphertext and the explicit part of the header is output to the insecure channel $\mathbf{IC}$.

The receiver converter dec_Π receives inputs from $\mathbf{IC}$ and queues the header-ciphertext pairs internally in a queue $\mathcal{Q}$. For each newly arrived message a notification is output at the outer interface. The next ciphertext C in the queue is decrypted if dec_Π is invoked with the implicit part of the corresponding header. The parameters for decryption are again the header as the additional data, the counter as the nonce and the shared key. On success, the corresponding plaintext is output at the outer interface. If decryption fails, the converter stops and signals an error by outputting $\perp$.

4.3 The Construction Notion

In order to show that the protocol $(\mathsf{enc}_\Pi, \mathsf{dec}_\Pi)$ constructs $\mathbf{ASC}$ from $[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ in the sense of constructive cryptography, we have to prove the *availability condition* and the *security condition* that are derived from the general construction notion in [16].

Random experiments. Both conditions make statements about random experiments $\mathbf{DR}$ in which a distinguisher $\mathbf{D}$ plays the role of an interactive environment for some resource $\mathbf{R}$. The distinguisher $\mathbf{D}$ is a system that provides inputs to the connected resource and receives the outputs generated by the resource. For example, $\mathbf{D}\big(\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B} [\mathbf{SK}_\mathcal{K}, \mathbf{IC}]\big)$ is the experiment that captures "the protocol in action" in the environment provided by $\mathbf{D}$. More concretely, in each step of these experiments, the distinguisher provides an input to one of the interfaces A, B, or E and observes the output that is generated in reaction to that input. This process continues iteratively by having $\mathbf{D}$ providing adaptively the next input and receiving the next output. The

---

[2]For example, the resource $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B} [\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ is obtained by attaching Alice's converter enc_Π at interface A and Bob's converter dec_Π at interface B of $[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$, where the interfaces are indicated by superscripts.

[3]Here, $(E, I) \in \mathcal{H}_\mathrm{E} \times \mathcal{H}_\mathrm{I}$ denotes an encoding of that pair as an element in $\mathcal{A}$. Abusing notation, we generally do not distinguish between a tuple and its encoding as an element in Σ^*.
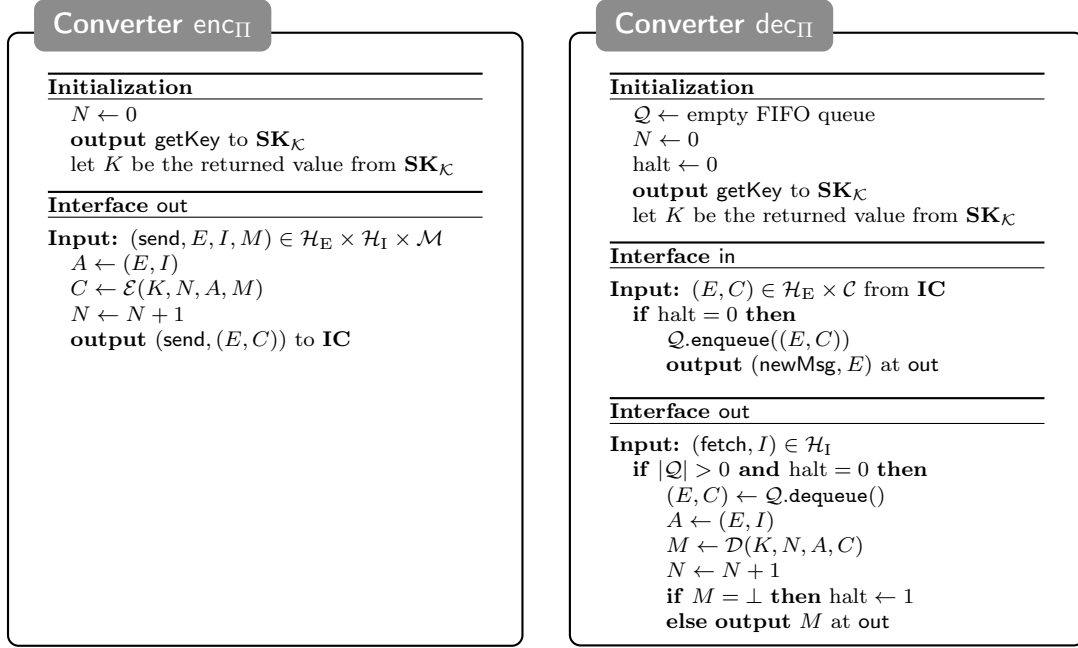
Figure 5: The protocol converters for the sender (left) and the receiver (right) that construct **ASC** via an AEAD-scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$.

experiment ends by $\mathbf{D}$ outputting a bit 0 or 1 that indicates its guess to which system it is connected. The *distinguishing advantage* of $\mathbf{D}$ for two resources $\mathbf{R}$ and $\mathbf{S}$ is defined as

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) = \Pr[\mathbf{DR} = 1] - \Pr[\mathbf{DS} = 1].$$

Availability condition. The first condition captures the situation when no attacker interferes with the protocol execution. We require that in this case, the intended functionality is available to the honest parties. This condition can be seen as a correctness condition for the protocol.

No attacker being present is formalized by a special converter dlv that is attached at interface E and always ensures the *delivery* of messages. Concretely, on any input at its inner interface, dlv outputs deliver to the channel connected to its inner interface and does not provide any service at its outer interface. Formally, the availability condition places a bound on the advantage in distinguishing the systems $\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}\mathsf{dlv}^{\mathsf{E}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ and $\mathsf{dlv}^{\mathsf{E}}\mathbf{ASC}$, i.e., a bound on

$$\Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}\mathsf{dlv}^{\mathsf{E}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}], \mathsf{dlv}^{\mathsf{E}}\mathbf{ASC}\right)$$
$$= \Pr\left[\mathbf{D}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}\mathsf{dlv}^{\mathsf{E}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]\right) = 1\right] - \Pr\left[\mathbf{D}\left(\mathsf{dlv}^{\mathsf{E}}\mathbf{ASC}\right) = 1\right]$$

for any distinguisher $\mathbf{D}$.

Security condition. The second condition captures the situation where an adversary attacks the protocol execution using its capabilities at interface E. The effects of such an attack have to be indistinguishable from the effects in the system corresponding

to the constructed resource with some simulator attached at the adversarial interface. This captures that all attacks on the protocol can be translated by a simulator to an attack on the constructed resource. Turned around, if the constructed resource is secure by definition, there is no successful attack on the protocol. More concretely, the security condition places a bound on the advantage in distinguishing the systems $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}]$ and $\mathsf{sim}_\mathrm{ASC}^\mathsf{E}\,\mathbf{ASC}$ for some simulator $\mathsf{sim}_\mathrm{ASC}$, i.e., a bound on

$$\Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}],\mathsf{sim}_\mathrm{ASC}^\mathsf{E}\,\mathbf{ASC}\right)$$
$$=\mathsf{Pr}\left[\mathbf{D}\left(\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}]\right)=1\right]-\mathsf{Pr}\left[\mathbf{D}\left(\mathsf{sim}_\mathrm{ASC}^\mathsf{E}\,\mathbf{ASC}\right)=1\right]$$

for any distinguisher $\mathbf{D}$.

4.4 Proof of the Construction

The following two lemmata relate the AEAD-security game to the distinguishing advantage in the availability and security condition, respectively. We begin with the availability condition:

Lemma 1. *There is an (efficient) transformation ρ described in the proof that maps distinguishers $\mathbf{D}$ for two resources to valid adversaries $\mathcal{A} = \rho(\mathbf{D})$ for the AEAD-security game such that*

$$\Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\mathsf{dlv}^\mathsf{E}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}],\mathsf{dlv}^\mathsf{E}\,\mathbf{ASC}\right)\leq\mathbf{Adv}_\Pi^{\mathrm{ae}}(\rho(\mathbf{D})).$$

Proof. In $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\mathsf{dlv}^\mathsf{E}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}]$, the converter dlv is attached at interface E and answers any output produced by $\mathbf{IC}$ with the input $\mathsf{deliver}$. This essentially converts $\mathbf{IC}$ into a reliable transmission channel: whatever pair (E, C) is input by converter enc_Π, it is immediately delivered to dec_Π that outputs a notification (newMsg, E) at its outer interface. Furthermore, if the ith input at interface A is $(\mathsf{send}, E_i, I_i, M_i)$, and the ith input at interface B is (fetch, I_i), then the output at interface B is M_i. The same holds for system $\mathsf{dlv}^\mathsf{E}\mathbf{ASC}$. Only if the ith input at interface B is (fetch, I_i') for $I_i' \neq I_i$, then the behavior of the two systems can differ: While $\mathsf{dlv}^\mathsf{E}\mathbf{ASC}$ always returns $\bot$ in this case, for $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\mathsf{dlv}^\mathsf{E}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}]$ it is possible that a message $M \neq \bot$ is returned. Since this is the only difference between the two systems, we can upper bound the distinguishing advantage by the probability that $\mathbf{D}$ can provoke such an output at interface B when interacting with $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\mathsf{dlv}^\mathsf{E}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}]$. It remains to bound the probability of this event, subsequently denoted by $\mathcal{F}$.

Note that $\mathcal{F}$ occurs exactly if the encryption algorithm of the AEAD-scheme returns a message $M \neq \bot$ on input a different additional data than used for encryption. Based on this observation, we build an adversary $\mathcal{A}$ that emulates a view towards distinguisher $\mathbf{D}$ that is identical to an interaction of $\mathbf{D}$ with $\mathsf{enc}_\Pi{}^\mathsf{A}\mathsf{dec}_\Pi{}^\mathsf{B}\mathsf{dlv}^\mathsf{E}\,[\mathbf{SK}_\mathcal{K},\mathbf{IC}]$ if $\mathcal{A}$ gets access to its real oracles. The probability of provoking event $\mathcal{F}$ is preserved in this case. In contrast, if $\mathcal{A}$ gets access to the ideal oracles, the condition for event $\mathcal{F}$ cannot be satisfied as we argue below. This is a suitable distinguishing criterion.

More formally, the reduction is defined as follows: The adversary $\mathcal{A} = \rho(\mathbf{D})$ initially sets $N_A, N_B \leftarrow 0$, initializes an empty FIFO queue $\mathcal{Q}$, and then emulates an execution to $\mathbf{D}$ as follows. When $\mathbf{D}$ inputs (send, E, I, M) at interface A, $\mathcal{A}$ ask the

```
Converter sim_ASC

Initialization                              Interface out
   Q₁, Q₂ ← empty FIFO queues               Input: deliver
   let K̂ ∈ K, N̂ ∈ N, Â ∈ A be arbitrary       if |Q₁| > 0 then
                                                  (E, C) ← Q₁.dequeue()
Interface in                                      execute commands for (inject, (E, C))

Input: (E, ℓ) ∈ H_E × ℕ
   choose M_ℓ ∈ M with |M_ℓ| = ℓ           Input: (inject, (E, C)) ∈ H_E × C
   C_ℓ ← E(K̂, N̂, Â, M_ℓ)                      if |Q₂| = 0 then
   C ←$ Σ^|C_ℓ|                                   output (injectStop, E) at in
   Q₁.enqueue((E, C))                        else
   Q₂.enqueue((E, C))                            (E', C') ← Q₂.dequeue()
   output (E, C) at out                          if E = E' and C = C' then
                                                     output deliver at in
                                                  else
                                                     output (injectStop, E) at in
```

Figure 6: The simulator for the security condition of the construction of **ASC**.

query $(N_A, (I, E), M)$ to the oracle Enc to receive the answer C. It then executes $N_A \leftarrow N_A + 1$ and Q.enqueue$((E, I, C))$, and emulates the output (newMsg, E) at interface B for **D**. Inputs (fetch, I) at interface B are ignored if Q is empty. Otherwise, $\mathcal{A}$ executes $(E', I', C') \leftarrow Q$.dequeue$()$ and asks the query $(N_B, (I, E'), C')$ to the oracle Dec to receive the answer M and sets $N_B \leftarrow N_B + 1$. It then emulates the output M at interface B for **D**.

If $M \neq \bot$ and $I' \neq I$ (i.e., the event $\mathcal{F}$ occurs), $\mathcal{A}$ stops and returns 1. If $M = \bot$, $\mathcal{A}$ ignores subsequent inputs at interface B. When **D** outputs a bit and $\mathcal{F}$ has not occurred, $\mathcal{A}$ returns 0. Observe that if $\mathcal{A}$ gets access to the ideal oracles, the conditions of event $\mathcal{F}$ cannot be met. We conclude the proof by noting that $\mathbf{Adv}_\Pi^{\mathrm{ae}}(\rho(\mathbf{D}))$ equals the probability of the event $\mathcal{F}$. □

The next lemma implies the security condition of the construction:

Lemma 2. *For the simulator* $\mathsf{sim}_{\mathrm{ASC}}$ *defined in Figure 6, there is an (efficient) transformation* ρ' *described in the proof that maps distinguishers* **D** *for two resources to valid adversaries* $\mathcal{A} = \rho'(\mathbf{D})$ *for the AEAD-security game such that*

$$\Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}], \mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\,\mathbf{ASC}\right) \leq \mathbf{Adv}_\Pi^{\mathrm{ae}}(\rho'(\mathbf{D})).$$

Proof. Let **D** be a distinguisher for $\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ and $\mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\mathbf{ASC}$. We define an adversary $\mathcal{A} = \rho'(\mathbf{D})$ for the AEAD-security game as follows. The adversary $\mathcal{A}$ initially sets $N_A, N_B, \mathtt{flag} \leftarrow 0$, initializes an empty FIFO queue $\mathcal{S}$ and two empty lists[4] $\mathcal{L}$ and $\mathcal{R}$, and then emulates an execution of **D** by translating inputs of the distinguisher to oracle queries as well as answers from the oracles to outputs of the resource for **D**. There are four types of inputs **D** can make:

(send, E, I, M) **at interface** A: If $\mathcal{R}$ contains strictly less than $N_A + 1$ elements, $\mathcal{A}$ asks the query $(N_A, (E, I), M)$ to the oracle Enc and receives the answer C. It then

---

[4]For a list L, we denote by $L \parallel x$ the list L with x appended. Furthermore, the ith element of a list L with n elements is denoted by $L[i]$ for $i \in \{0, \dots, n-1\}$.

stores (N_A, E, I, M, C) in the list $\mathcal{L}$, emulates the output (E, C) at interface E for $\mathbf{D}$, sets $N_A \leftarrow N_A + 1$, and executes $\mathcal{S}.\text{enqueue}((E, C))$.

If $\mathcal{R}$ contains at least $N_A + 1$ elements, there is a pair $\mathcal{R}[N_A] = (E, C)$. $\mathcal{A}$ asks the query $(N_A, (E, I), C)$ to the oracle Dec to receive the plaintext M. If $M \neq \perp$, $\mathcal{A}$ sets flag $\leftarrow 1$, returns 1 as its decision and halts. If $M = \perp$, the tuple $(N_A, E, I, \perp, C)$ is stored in $\mathcal{L}$ and $\mathcal{A}$ asks the query $(N_A, (E, I), M)$ to the oracle Enc, receives the answer C and stores (N_A, E, I, M, C) in the list $\mathcal{L}$. Finally, $\mathcal{A}$ emulates the output (E, C) at interface E for $\mathbf{D}$, sets $N_A \leftarrow N_A + 1$, and executes $\mathcal{S}.\text{enqueue}((E, C))$.

deliver **at interface** E: If $|\mathcal{S}| > 0$, $\mathcal{A}$ executes $(E, C) \leftarrow \mathcal{S}.\text{dequeue}()$ followed by $\mathcal{R} \leftarrow \mathcal{R} \parallel (E, C)$. If $\perp$ has not been output at interface B, $\mathcal{A}$ emulates the output (newMsg, E) at interface B.

$(\mathsf{inject}, (E, C))$ **at interface** E: The adversary $\mathcal{A}$ executes $\mathcal{R} \leftarrow \mathcal{R} \parallel (E, C)$. If $\perp$ has not been output at interface B, $\mathcal{A}$ emulates the output (newMsg, E) at interface B.

(fetch, I) **at interface** B: If $\mathcal{R}$ is empty, the input is ignored. Otherwise, $\mathcal{A}$ executes $(E, C) \leftarrow \mathcal{R}[N_B]$. If $(N_B, E, I, \perp, C)$ is in $\mathcal{L}$, $\mathcal{A}$ emulates the output $\perp$ at interface B and ignores subsequent inputs at interface B. If (N_B, E, I, M, C) is in $\mathcal{L}$ for some $M \in \mathcal{M}$, $\mathcal{A}$ emulates the output M at interface B for $\mathbf{D}$ and sets $N_B \leftarrow N_B + 1$. Otherwise, $\mathcal{A}$ asks the query $(N_B, (E, I), C)$ to the oracle Dec to receive the plaintext M. The output M is emulated at interface B and the counter N_B is incremented. If $M = \perp$, $\mathcal{A}$ ignores subsequent inputs at interface B.

When $\mathbf{D}$ outputs a bit b and if flag $= 0$, $\mathcal{A}$ returns b and halts. Note that $\mathcal{A}$ is a valid adversary since it asks at most one Enc and Dec query for each nonce (and therefore does not repeat queries) and never asks a query to the oracle Dec for a ciphertext that has been returned by a query to Enc for the same parameters (because for such ciphertext, the corresponding tuple is in the list $\mathcal{L}$). To analyze the success probability of $\mathcal{A}$, let F be the random variable that takes on the value of flag at the end of the random experiment between $\mathcal{A}$ and $\mathbf{Real}_\Pi$.

We claim that the view of $\mathbf{D}$ when connected to $\mathsf{sim}^{\mathsf{E}}_{\mathrm{ASC}} \mathbf{ASC}$ is identical to the view emulated by $\mathcal{A}$ with access to the ideal oracles. Additionally, the view of $\mathbf{D}$ when connected to $\mathsf{enc}_\Pi{}^{\mathsf{A}} \mathsf{dec}_\Pi{}^{\mathsf{B}} [\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]$ is identical to the view emulated by $\mathcal{A}$ with access to the real oracles as long as flag $= 0$. This implies the statement of the lemma:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ae}}_\Pi(\mathcal{A}) &= \Pr[\mathcal{A}^{\mathbf{Real}_\Pi} = 1] - \Pr[\mathcal{A}^{\mathbf{Ideal}_\Pi} = 1] \\
&= \Pr[F = 1] + \Pr[F = 0] \cdot \underbrace{\Pr[\mathcal{A}^{\mathbf{Real}_\Pi} = 1 \mid F = 0]}_{= \Pr[\mathbf{D}(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]) = 1]} - \Pr[\mathcal{A}^{\mathbf{Ideal}_\Pi} = 1] \\
&\geq \Pr\left[\mathbf{D}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}]\right) = 1\right] - \Pr\left[\mathbf{D}\left(\mathsf{sim}^{\mathsf{E}}_{\mathrm{ASC}} \mathbf{ASC}\right) = 1\right] \\
&= \Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}}, \mathbf{IC}], \mathsf{sim}^{\mathsf{E}}_{\mathrm{ASC}} \mathbf{ASC}\right).
\end{aligned}$$

A proof of this claim can be found in Appendix A. It involves the analysis of the same possible possible inputs by $\mathbf{D}$ as above and comparing the resulting outputs. $\square$

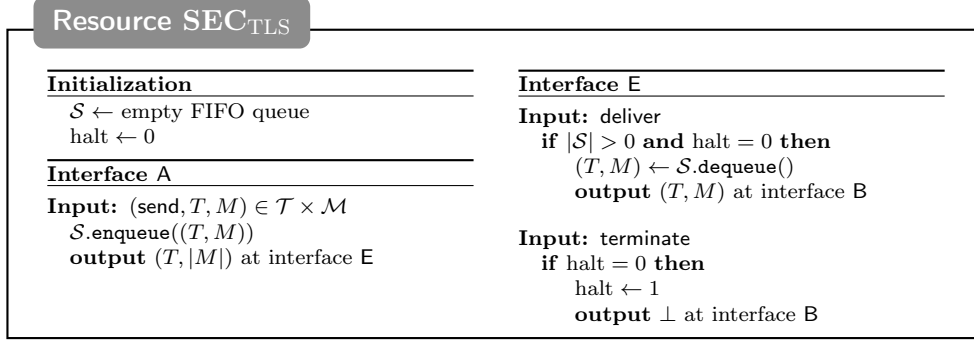The following theorem summarizes the results from Lemma 1 and Lemma 2.

Resource SEC_TLS

Initialization
  $\mathcal{S} \leftarrow$ empty FIFO queue
  halt $\leftarrow 0$

Interface A
Input: $(\mathsf{send}, T, M) \in \mathcal{T} \times \mathcal{M}$
  $\mathcal{S}.\mathsf{enqueue}((T, M))$
  output $(T, |M|)$ at interface E

Interface E
Input: deliver
  if $|\mathcal{S}| > 0$ **and** halt $= 0$ **then**
    $(T, M) \leftarrow \mathcal{S}.\mathsf{dequeue}()$
    output (T, M) at interface B

Input: terminate
  if halt $= 0$ **then**
    halt $\leftarrow 1$
    output $\perp$ at interface B

Figure 7: Description of the channel $\mathbf{SEC}_{\mathrm{TLS}}$.

Theorem 1. *The protocol* $(\mathsf{enc}_\Pi, \mathsf{dec}_\Pi)$ *constructs* $\mathbf{ASC}$ *from* $[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$. *More specifically, we have for the simulator* $\mathsf{sim}_{\mathrm{ASC}}$ *in Figure 6 and for all distinguishers* $\mathbf{D}$

$$\Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}\mathsf{dlv}^{\mathsf{E}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}], \mathsf{dlv}^{\mathsf{E}}\,\mathbf{ASC}\right) \leq \mathbf{Adv}_\Pi^{\mathrm{ae}}(\rho(\mathbf{D}))$$

$$and \qquad \Delta^{\mathbf{D}}\left(\mathsf{enc}_\Pi{}^{\mathsf{A}}\mathsf{dec}_\Pi{}^{\mathsf{B}}[\mathbf{SK}_\mathcal{K}, \mathbf{IC}], \mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\,\mathbf{ASC}\right) \leq \mathbf{Adv}_\Pi^{\mathrm{ae}}(\rho'(\mathbf{D})),$$

where ρ *and* ρ' *are the reductions defined in the proofs of Lemma 1 and Lemma 2, respectively.*

5 The Goal and Improvement of TLS 1.3

Version 1.3 of TLS is currently in draft state at the IETF.[5] Unlike TLS 1.2, the new version of the record payload protection protocol mandates AEAD-ciphers[6] and the format of the authenticated data has changed. More specifically, in the current draft, the authenticated data in TLS 1.3 consists of the sequence number of the current fragment, the protocol version, and the message type, e.g., whether it is an "alert," a "handshake," or an "application" message. This section treats the record protocol and assumes that a shared key has been derived in the TLS handshake.

5.1 Formalizing the Goal of TLS Record Payload Protection

What does it mean for the TLS record layer to be secure? We propose a simple answer to this question in the form of a new channel abstraction. Each packet in the TLS record layer contains a payload and specifies its associated type. While the entire packet is authenticated, only the content of the packet has to be private and hidden from the attacker. This resembles a specific type of channel: a secure channel where messages are tagged with a non-private type-flag. The TLS record payload protection can be considered secure if it provably constructs this secure channel. We formalize this channel as the resource $\mathbf{SEC}_{\mathrm{TLS}}$ and provide a formal description thereof in Figure 7. The set of types is defined as $\mathcal{T} := \{0, \dots, 255\}$.

---

[5]We refer to the most recent draft (retrieved on April 6, 2015) that is available for download at `https://tools.ietf.org/html/draft-ietf-tls-tls13-05`.
[6]Previous versions of TLS supported MAC-then-Encrypt with different encryption modes.

```
┌─ Converter tlsSnd ──────────────────┐        ┌─ Converter tlsRcv ──────────────────┐
│                                     │        │                                     │
│  Initialization                     │        │  Initialization                     │
│  ───────────────                    │        │  ───────────────                    │
│     V ← {3, 4}                      │        │     V ← {3, 4}                      │
│                                     │        │                                     │
│  Interface out                      │        │  Interface in                       │
│  ───────────────                    │        │  ───────────────                    │
│  Input: (send, T, M) ∈ 𝒯 × ℳ       │        │  Input: (newMsg, T) ∈ 𝒯             │
│     output (send, T, V, M) to ASC   │        │     if halt = 0 then                │
│                                     │        │        output (fetch, V) to ASC     │
│                                     │        │        let M be the returned value  │
│                                     │        │           from ASC                  │
│                                     │        │        if M ≠ ⊥ then                 │
│                                     │        │           output (T, M) at out      │
│                                     │        │        else                         │
│                                     │        │           halt ← 1                  │
│                                     │        │           output ⊥ at out           │
│                                     │        │                                     │
└─────────────────────────────────────┘        └─────────────────────────────────────┘
```
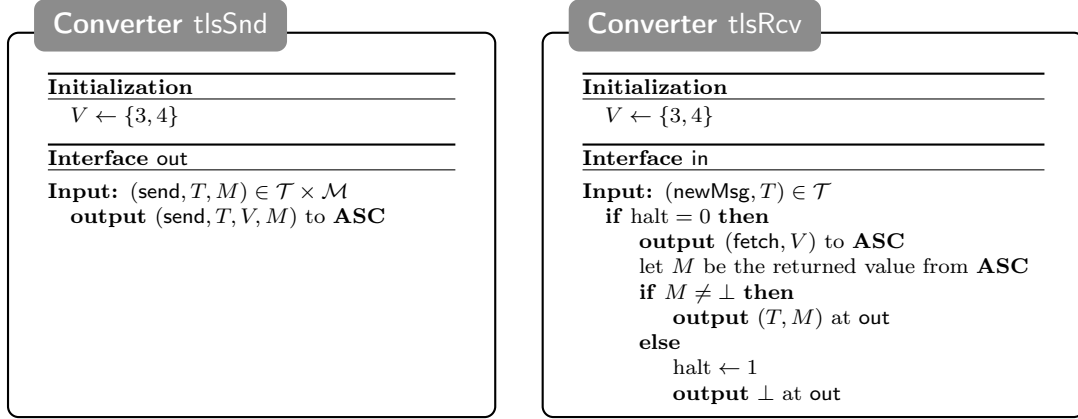
Figure 8: The protocol converters for the sender (left) and the receiver (right) that construct $\mathbf{SEC}_{\mathrm{TLS}}$ from $\mathbf{ASC}$.

Note that in contrast to $\mathbf{ASC}$, the channel $\mathbf{SEC}_{\mathrm{TLS}}$ does not contain an implicit part of the header and messages are directly delivered to Bob without the need to fetch them. Therefore, $\mathbf{SEC}_{\mathrm{TLS}}$ does not allow the authentication of data without sending it. One can thus view $\mathbf{SEC}_{\mathrm{TLS}}$ as an augmented secure channel that is more restricted than $\mathbf{ASC}$ but also simpler to use.

5.2   Achieving the Goal

In this section, we present a construction of the channel $\mathbf{SEC}_{\mathrm{TLS}}$ from $\mathbf{ASC}$. To this end, we introduce the protocol (tlsSnd, tlsRcv), which is described in Figure 8 and manages the usage of the resource $\mathbf{ASC}$. We adhere closely to the current TLS specification in setting the corresponding values. The implicit part I of the header consists of the protocol version V (which corresponds to $\{3, 4\}$ and consists of two bytes[7]) and the explicit part E consists of the message type T (one byte).

Theorem 2. *The protocol (tlsSnd, tlsRcv) constructs* $\mathbf{SEC}_{\mathrm{TLS}}$ *from* $\mathbf{ASC}$. *More specifically, we have for the simulator* $\mathsf{sim}_{\mathrm{TLS}}$ *defined in Figure 9 and for all distinguishers* $\mathbf{D}$

$$\Delta^{\mathbf{D}}\left(\mathsf{tlsSnd}^{\mathsf{A}}\mathsf{tlsRcv}^{\mathsf{B}}\mathsf{dlv}^{\mathsf{E}}\mathbf{ASC}, \mathsf{dlv}^{\mathsf{E}}\mathbf{SEC}_{\mathrm{TLS}}\right) = 0 \tag{1}$$

$$and \quad \Delta^{\mathbf{D}}\left(\mathsf{tlsSnd}^{\mathsf{A}}\mathsf{tlsRcv}^{\mathsf{B}}\mathbf{ASC}, \mathsf{sim}_{\mathrm{TLS}}^{\mathsf{E}}\mathbf{SEC}_{\mathrm{TLS}}\right) = 0. \tag{2}$$

Proof. The availability condition (1) is easy to verify: On input (send, T, M) at interface A, the system $\mathsf{dlv}^{\mathsf{E}}\mathbf{SEC}_{\mathrm{TLS}}$ directly outputs (T, M) at interface B. The same holds for system $\mathsf{tlsSnd}^{\mathsf{A}}\mathsf{tlsRcv}^{\mathsf{B}}\mathsf{dlv}^{\mathsf{E}}\mathbf{ASC}$: On input (send, T, M), the converter tlsSnd inputs (send, T, V, M) to $\mathbf{ASC}$. The converter tlsRcv then obtains the notification (newMsg, T) and queries (fetch, V) to $\mathbf{ASC}$, which results in the output M from $\mathbf{ASC}$, which in turn triggers tlsRcv to output (T, M). Since the two systems behave identically, every distinguisher has advantage 0 in distinguishing them, i.e., (1) follows.

---

[7]The value $\{3, 4\}$ corresponds to TLS version 1.3. The reason for this value is that the version of TLS 1.0, as the successor of SSL 3.0, is encoded as the value $\{3, 1\}$.
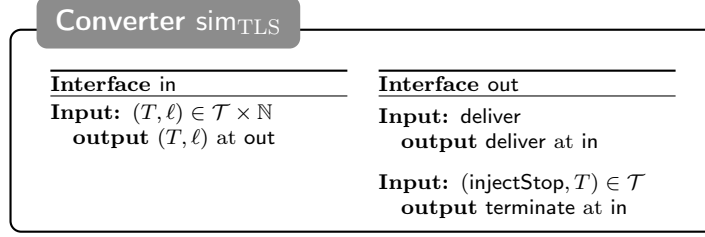
Figure 9: The simulator for the security condition of the construction of $\mathbf{SEC}_{\mathrm{TLS}}$.

To verify the security condition (2), we distinguish the possible inputs to the system:

Input (send, T, M) **at interface** A: In the system $\mathsf{tlsSnd}^{\mathsf{A}}\mathsf{tlsRcv}^{\mathsf{B}}\mathbf{ASC}$, this input results in the converter tlsSnd inputting (send, T, V, M) to $\mathbf{ASC}$, which yields the output $(T, |M|)$ at interface E of $\mathbf{ASC}$. In $\mathsf{sim}_{\mathrm{TLS}}^{\mathsf{E}}\mathbf{SEC}_{\mathrm{TLS}}$, the values $(T, |M|)$ are given to the simulator, which then outputs $(T, |M|)$ at its outer interface.

Input $\mathsf{deliver}$ **at interface** E: In $\mathsf{tlsSnd}^{\mathsf{A}}\mathsf{tlsRcv}^{\mathsf{B}}\mathbf{ASC}$, if the queue $\mathcal{S}$ in $\mathbf{ASC}$ is empty, nothing happens. Otherwise, tlsRcv receives the notification (newMsg, T). Then, tlsRcv inputs (fetch, V) to $\mathbf{ASC}$ if it has not already halted. In this case, there have been only inputs $\mathsf{deliver}$ at interface E and therefore the verification within $\mathbf{ASC}$ succeeds. Thus, tlsRcv obtains the message M and outputs (T, M).

In $\mathsf{sim}_{\mathrm{TLS}}^{\mathsf{E}}\mathbf{SEC}_{\mathrm{TLS}}$, the simulator inputs $\mathsf{deliver}$ to $\mathbf{SEC}_{\mathrm{TLS}}$. If $\mathcal{S}$ in $\mathbf{SEC}_{\mathrm{TLS}}$ is empty, nothing happens. Otherwise, the next tuple (T, M) in $\mathcal{S}$ is output at interface B if the channel has not halted before.

Input $(\mathsf{injectStop}, T)$ **at interface** E: In the system $\mathsf{tlsSnd}^{\mathsf{A}}\mathsf{tlsRcv}^{\mathsf{B}}\mathbf{ASC}$, the notification (newMsg, T) is output to tlsRcv. The converter tlsRcv then outputs (fetch, V) to $\mathbf{ASC}$ and since the element is an inserted empty element, the verification within $\mathbf{ASC}$ fails and tlsRcv outputs $\bot$ and stops by setting $\mathsf{halt} \leftarrow 1$.

In $\mathsf{sim}_{\mathrm{TLS}}^{\mathsf{E}}\mathbf{SEC}_{\mathrm{TLS}}$, $\mathsf{sim}_{\mathrm{TLS}}$ terminates the session, which causes the output $\bot$ at interface B and results in no further messages being processed by Bob.

To see that the two described systems behave identically, we only have to observe that they both terminate the session if an empty message is injected into the channel and that all inputs are delivered in order until termination. We again conclude that every distinguisher has advantage 0 in distinguishing these systems, i.e., we obtain (2). This completes the proof. □

5.3 Deriving a More Efficient Proposal for TLS 1.3

We have shown that $(\mathsf{tlsSnd}, \mathsf{tlsRcv})$ constructs $\mathbf{SEC}_{\mathrm{TLS}}$ from $\mathbf{ASC}$. Since by Theorem 1, the protocol $(\mathsf{enc}_{\Pi}, \mathsf{dec}_{\Pi})$ constructs the channel $\mathbf{ASC}$ from a shared secret key and an insecure channel, we can invoke the composition theorem of constructive cryptography to conclude that the composition of both protocols constructs $\mathbf{SEC}_{\mathrm{TLS}}$ from a shared key $\mathbf{SK}_{\mathcal{K}}$ and an insecure channel $\mathbf{IC}$, see Figure 10 for a graphical illustration of the composed protocol.

The composed protocol for the sender $\mathsf{tls}_{\mathsf{A}}$ works as follows: On input (send, T, M), the message M is encrypted with a call to the AEAD-scheme as $C \leftarrow \mathcal{E}(K, N, A, M)$,
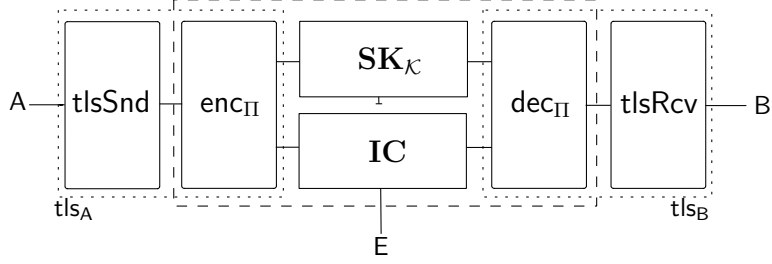
Figure 10: Illustration of the composed protocol $(\mathsf{tls_A}, \mathsf{tls_B})$. For a secure AEAD-scheme, the resource $\mathsf{enc_\Pi}^\mathsf{A}\mathsf{dec_\Pi}^\mathsf{B}\,[\mathbf{SK}_\mathcal{K}, \mathbf{IC}]$ inside the dashed box is indistinguishable from $\mathsf{sim}^\mathsf{E}_{\mathrm{ASC}}\mathbf{ASC}$.

where K is the shared key retrieved from $\mathbf{SK}_\mathcal{K}$, N is the internal counter and $A = (T, V)$ is the additional data. Finally, the pair (T, C) is sent over the insecure channel.

The protocol for the receiver $\mathsf{tls_B}$ works analogously: On input a new pair (T, C) from $\mathbf{IC}$, the ciphertext is decrypted to $M \leftarrow \mathcal{D}(K, N, A, C)$, where N is the internal counter, A is the additional data and K is the shared key as above. Note that the implicit part of the header is fixed and provided by tlsRcv immediately after receiving the notification (newMsg, T) from $\mathsf{dec_\Pi}$.

In summary, the protocol $(\mathsf{tls_A}, \mathsf{tls_B})$ provably achieves the goal of the TLS record layer and is more efficient than existing proposals as explained below. Note that the key resource $\mathbf{SK}_\mathcal{K}$ is constructed by the handshake protocol if both parties are authenticated. In [14], the authors consider the more general case where only one party is authenticated, which yields a weaker key resource. We have chosen the setting where both sides are authenticated to simplify the presentation, but we point out that our result can be generalized to the more general setting straightforwardly.

Our proposal. Based on the above construction, we derive improvements to the current draft of TLS 1.3. More concretely, we recommend:

1. The nonce of the AEAD-scheme should be a counter (i.e., the sequence number of the fragment) and not be explicitly sent as part of the TLS record. The nonce can be left-padded with zeros to be of the appropriate length.

2. The sequence number should be removed from the additional data part.

3. After the handshake, the version number does not need to be transmitted explicitly as part of the TLS record. However, it should still be part of the additional data.

Our proposal improves upon current drafts of TLS 1.3 by reducing the amount of data sent in each TLS record. It further reduces the number of elements in the additional data. Moreover, our proposal comes with a rigorous security proof. Because of this proof, it is guaranteed that our choice of parameters is adequate. For example, this clarifies that the nonce need not be unpredictable or derived from other values, which is a priori unclear.

We are aware that item 3 would require a new structure of TLS fragments and hence there might be objections against this change. However, we stress that only respecting item 1 and item 2 of our proposal is also secure (i.e., constructs $\mathbf{SEC}_{\mathrm{TLS}}$). The proof

for the case where the version number is moved to the explicit part of the header is essentially identical to the one presented in Section 5.2.

References

[1] Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Inf. Comput. 205(12), 1685–1720 (Dec 2007)

[2] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Advances in Cryptology – ASIACRYPT 2000, pp. 531–545. Springer Berlin Heidelberg (2000)

[3] Bellare, M., Rogaway, P.: Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In: Advances in Cryptology – ASIACRYPT 2000, pp. 317–330. Springer (2000)

[4] Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In: Fast Software Encryption, pp. 389–407. Springer (2004)

[5] Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145. IEEE (2001)

[6] Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Advances in Cryptology – CRYPTO 2003, pp. 565–582. Springer (2003)

[7] Coretti, S., Maurer, U., Tackmann, B.: Constructing confidential channels from authenticated channels – public-key encryption revisited. In: Advances in Cryptology – ASIACRYPT 2013, Part I, pp. 134–153. Springer (2013)

[8] Gajek, S., Manulis, M., Pereira, O., Sadeghi, A., Schwenk, J.: Universally composable security analysis of TLS. In: Proceedings of ProvSec 2008, pp. 313–327 (2008)

[9] He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.: A modular correctness proof of IEEE 802.11i and TLS. In: Proceedings of the ACM Conference on Computer and Communications Security (ACM CCS'05), pp. 2–15 (2005)

[10] Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption: AEZ and the problem that it solves. In: Advances in Cryptology – EUROCRYPT 2015, pp. 15–44. Springer (2015)

[11] Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the Security of TLS-DHE in the Standard Model. In: Advances in Cryptology – CRYPTO 2012, pp. 273–293. Springer (2012)

[12] Jutla, C.: Encryption modes with almost free message integrity. In: Advances in Cryptology – EUROCRYPT 2001, pp. 529–544. Springer (2001)

[13] Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Fast Software Encryption, pp. 284–299. Springer (2001)

[14] Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: (De-)Constructing TLS. Cryptology ePrint Archive, Report 2014/020 (2014)

[15] Krawczyk, H., Paterson, K., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Proceedings of CRYPTO 2013, pp. 429–448 (2013)

[16] Maurer, U.: Constructive cryptography – a new paradigm for security definitions and proofs. In: Mödersheim, S., Palamidessi, C. (eds.) Theory of Security and Applications, Lecture Notes in Computer Science, vol. 6993, pp. 33–56. Springer (2012)

[17] Maurer, U., Renner, R.: Abstract cryptography. In: Chazelle, B. (ed.) The Second Symposium on Innovations in Computer Science, ICS 2011, pp. 1–21. Tsinghua University Press (2011)

[18] Maurer, U., Rüedlinger, A., Tackmann, B.: Confidentiality and integrity: a constructive perspective. In: TCC 2012, pp. 209–229. Springer (2012)

[19] Morrissey, P., Smart, N., Warinschi, B.: A modular security analysis of the TLS handshake protocol. In: Proceedings of ASIACRYPT 2008, pp. 55–73 (2008)

[20] Paterson, K, Ristenpart, T., Shrimpton, T.: Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol. In: Advances in Cryptology — ASIACRYPT 2011, pp. 372–389 (2011)

[21] Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp. 184–200. IEEE Computer Society (2001)

[22] Rogaway, P.: Authenticated-encryption with associated-data. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 98–107. ACM (2002)

[23] Rogaway, P., Bellare, M., Black, J.: OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Transactions on Information and System Security (TISSEC) 6(3), 365–403 (2003)

[24] Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Advances in Cryptology – EUROCRYPT 2006, pp. 373–390. Springer (2006)

[25] Wagner D., Schneier, B.: Analysis of the SSL 3.0 protocol. In: USENIX – Workshop on Electronic Commerce, pp. 29–40 (1996)

A Concluding the Proof of Lemma 2

To conclude the proof of Lemma 2 in Section 4.4, we have to show that the view of $\mathbf{D}$ when connected to $\mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\mathbf{ASC}$ is identical to the view emulated by $\mathcal{A}$ with access to the ideal oracles and the view of $\mathbf{D}$ when connected to $\mathsf{enc}_{\Pi}^{\mathsf{A}}\mathsf{dec}_{\Pi}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}},\mathbf{IC}]$ is identical to the view emulated by $\mathcal{A}$ with access to the real oracles as long as $\mathtt{flag}=0$.

To prove this claim, we distinguish the possible inputs by $\mathbf{D}$ and compare the resulting outputs:

(send, E, I, M) **at interface** A: In system $\mathsf{enc}_{\Pi}^{\mathsf{A}}\mathsf{dec}_{\Pi}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}},\mathbf{IC}]$, the converter enc_{Π} evaluates $C \leftarrow \mathcal{E}(K, N, (E, I), M)$, where N is the number of sent messages before this input. The explicit part E of the header is sent together with C over $\mathbf{IC}$, which outputs the pair (E, C) at interface E. The same output is emulated by $\mathcal{A}$ in the real game since the oracle Enc in this case also evaluates the algorithm $\mathcal{E}$.

In the system $\mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\mathbf{ASC}$, the triple (E, I, M) is inserted into the senders queue of $\mathbf{ASC}$ and the pair $(E, |M|)$ is output to the simulator $\mathsf{sim}_{\mathrm{ASC}}$, which in turn generates a uniformly random ciphertext C of the same length as ciphertexts for M. Note that by Definition 1, the length of ciphertexts only depend on the length of the message, so the values of $\hat{K}$, $\hat{N}$, and $\hat{A}$ used by $\mathsf{sim}_{\mathrm{ASC}}$ to determine this length are irrelevant. The simulator then stores (E, C) in its own queue for later reference and outputs this pair at interface E. Note that Enc in $\mathbf{Ideal}_{\Pi}$ generates ciphertexts with the same distribution, so the view emulated by $\mathcal{A}$ is identical.

$\mathsf{deliver}$ **at interface** E: If the sender's queue it non-empty, the next element (E, C) is dequeued from it and $\mathbf{D}$ receives the output (newMsg, E) from interface B if there has not been an output $\bot$ in both systems and in the emulated view.

$(\mathsf{inject}, (E, C))$ **at interface** E: In $\mathsf{enc}_{\Pi}^{\mathsf{A}}\mathsf{dec}_{\Pi}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}},\mathbf{IC}]$, the injected pair is inserted into the receiver's queue of the converter dec_{Π} and the notification (newMsg, E) is output at interface B.

In $\mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\mathbf{ASC}$, the simulator checks whether the injected pair is equal to the top-element (E', C') of its queue $\mathcal{Q}_2$. If this is the case, the simulator outputs $\mathsf{deliver}$ with the effect that the notification (newMsg, E) is output at interface B. If $(E, C) \neq (E', C')$, $\mathsf{sim}_{\mathrm{ASC}}$ injects a stop element by $(\mathsf{injectStop}, E)$, which also yields the output (newMsg, E) at interface B. Note that by definition of $\mathbf{ASC}$, this element is guaranteed to yield $\bot$ when fetched at interface B.

We see that in the emulation by $\mathcal{A}$, $\mathbf{D}$ receives (newMsg, E) from interface B if there has not been an output $\bot$ before at interface B. The same holds for both systems $\mathsf{enc}_{\Pi}^{\mathsf{A}}\mathsf{dec}_{\Pi}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}},\mathbf{IC}]$ and $\mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}\mathbf{ASC}$ in an interaction with $\mathbf{D}$.

(fetch, I) **at interface** B: Assume this is the ith input at interface B, there have been at least i inputs $\mathsf{deliver}$ or inject at interface E, and there has not been an output $\bot$ so far (otherwise the input is always ignored). In the system $\mathsf{enc}_{\Pi}^{\mathsf{A}}\mathsf{dec}_{\Pi}^{\mathsf{B}}[\mathbf{SK}_{\mathcal{K}},\mathbf{IC}]$, the converter dec_{Π} retrieves the top element of its queue. This value is equal to the ith delivered or injected pair (E, C) at interface E. The converter dec_{Π} then computes $M \leftarrow \mathcal{D}(K, i-1, (E, I), C)$ and outputs M.

In the view emulated by $\mathcal{A}$ in the real game, (E, C) also corresponds to the ith delivered or injected pair. If the tuple $(i-1, E, I, M, C)$ is found in $\mathcal{L}$ for some

$M \in \mathcal{M}$, M is output at interface B. By the correctness of the AEAD-scheme, M is then equal to the output of the algorithm $\mathcal{D}$ for the corresponding parameters. If no such tuple is in $\mathcal{L}$, $\mathcal{A}$ decrypts C with the corresponding parameters using the oracle Dec and also outputs the resulting message at interface B. Since the real oracle Dec evaluates $\mathcal{D}$, we conclude that the views are identical in this case.

In $\mathsf{sim}_{\mathrm{ASC}}^{\mathsf{E}}$ **ASC**, the resource checks whether $I = I'$, where (E', I', M') is the next element in the queue $\mathcal{R}$. If this is the case, it outputs M' at interface B, otherwise it outputs $\perp$. Furthermore, only those elements can be successfully fetched that do not correspond to stop-elements $(\perp, \perp, \perp)$. By construction of the simulator, the ith element of the sender's queue is only delivered if the ith injected pair (E, C) at interface E matches the simulated pair output at interface E in reaction to the ith input $(\mathsf{send}, E', I', M')$ at interface A. In any other case, a stop-element is injected into the receiver's queue.

To determine whether the values of the ith injection match the simulated values for the ith input at interface A, $\mathsf{sim}_{\mathrm{ASC}}$ maintains the queue $\mathcal{Q}_2$ such that its top element, after i injections, stores exactly these values. Note that the queue $\mathcal{Q}_1$ on the other hand is only needed to simulate the queue of the insecure channel **IC** in the real world and to figure out the next message in the simulation of a deliver-request.

In the view emulated by $\mathcal{A}$ in the ideal game, the list $\mathcal{L}$ ensures that the same message M' is output at interface B if all the values match as above. Furthermore, if there is not a match, the output is $\perp$ because the ideal oracle Dec always returns $\perp$. In particular, the condition that the ith simulated pair correspond to the ith injected pair is equivalent to requiring that the tuple $(i-1, E, I, M, C)$, for some message M, is an element of $\mathcal{L}$.

Hence, the views for **D** are also identical in this case.

This concludes the proof of the claim and thus of the lemma.