A preliminary version of this paper appears in the proceedings of the *35th Annual Cryptology Conference (CRYPTO 2015)*, DOI: 10.1007/978-3-662-48000-7_27. This is the full version.

Data Is a Stream: Security of Stream-Based Channels

Marc Fischlin[1] Felix Günther[1] Giorgia Azzurra Marson[2] Kenneth G. Paterson[3]

[1] Cryptoplexity, Technische Universität Darmstadt, Germany
[2] NEC Laboratories Europe, Germany
[3] Information Security Group, Royal Holloway, University of London, U.K.
`marc.fischlin@cryptoplexity.de, guenther@cs.tu-darmstadt.de, giorgia.marson@neclab.eu, kenny.paterson@rhul.ac.uk`

December 8, 2017

Abstract. The common approach to defining secure channels in the literature is to consider transportation of discrete messages provided via atomic encryption and decryption interfaces. This, however, ignores that many practical protocols (including TLS, SSH, and QUIC) offer streaming interfaces instead, moreover with the complexity that the network (possibly under adversarial control) may deliver arbitrary fragments of ciphertexts to the receiver. To address this deficiency, we initiate the study of stream-based channels and their security. We present notions of confidentiality and integrity for such channels, akin to the notions for atomic channels, but taking the peculiarities of streams into account. We provide a composition result for our setting, saying that combining chosen-plaintext confidentiality with integrity of the transmitted ciphertext stream lifts confidentiality of the channel to chosen-ciphertext security. Notably, for our proof of this theorem in the streaming setting we need an additional property, called error predictability. We give an AEAD-based construction that achieves our notion of a secure stream-based channel. The construction matches rather well the one used in TLS, providing validation of that protocol's design. Finally, we study how applications that actually aim at transporting atomic messages can do so safely over a stream-based channel. We provide corresponding security notions and a generic and secure 'encode-then-stream' paradigm.

Keywords. Secure channel, data stream, AEAD, confidentiality, integrity, fragmentation

Contents

1 Introduction

The most widely-used application for cryptography today is still secure communications—providing a *secure channel* for the transmission of data between two parties. Secure channel protocols are numerous and diverse in their features, operating at different network layers and offering different security services. Prominent examples can be found in GSM, UMTS and LTE [rGPP] mobile telecommunications systems, in WEP, WPA and WPA2 [oEEE] (which secure wireless LAN communications), IPsec [KS05] (which provides security at the IP layer), TLS [DR08] and DTLS [RM12] (which run over TCP [Pos81] and UDP [Pos80], respectively), Google's QUIC protocol [QUI], and SSH [YL06a] (an 'application layer' secure protocol).

AEAD and secure channels in the literature. Authenticated Encryption with Associated Data (AEAD) [Rog02] has emerged as being the right cryptographic tool for building secure channels. AEAD provides both confidentiality and integrity guarantees for data. However, on its own, AEAD does not constitute a secure channel. For example, in most practical situations, a secure channel should provide more than simple encryption of messages, but also guarantee detection of (and possibly recovery from) out-of-order delivery and replays of messages. Furthermore, a secure channel should deal with error handling, with errors potentially arising from both cryptographic and non-cryptographic processing—whether or not to tear-down a secure channel session if an error is encountered, and how (and indeed whether) to signal errors to the other side. As another difference, some secure channel designs (such as IPsec and to a limited extent TLS) have additional features that can be used to provide protection against traffic analysis. A secure channel may accept messages of arbitrary length and need to fragment these before encryption, and may reassemble these fragments again after decryption; alternatively, it may present to applications a maximum message size that is well-matched to the underlying network infrastructure. Finally, and most importantly in the context of the paper here, a secure channel may be designed to protect a *stream* of data rather than the series of discrete messages that is usually found in cryptographic abstractions.

There is, then, a substantial gap between what the AEAD primitive can reasonably provide and the needs of secure channels. We are not the first to recognize this gap, of course. For example, Bellare et al. [BKN02, BKN04] extended the standard security notions of confidentiality and integrity for symmetric encryption to the stateful setting, enabling the treatment of security of the ordering of discrete messages in a secure channel, with application to the analysis of SSH being their principle motivation. Their notions were later extended by Kohno et al. [KPB03] to include a richer variety of features, suitable for handling channels that permit (or deny) replays, message drops, and reordering. Meanwhile, Namprempre [Nam02] gave a characterization of channels secure in the UC sense (as introduced by Canetti and Krawczyk [CK01]) in terms of standard, game-based notions of security for AEAD. Even earlier, Shoup [Sho99] introduced a basic functional model for secure channels in a simulation-based setting; a secure channel concept was established by Canetti in the UC framework in [Can00].

More recently, Maurer and Tackmann [MT10] developed the constructive cryptography framework and applied it to the study of generic compositions with encryption and authentication with a view to achieving a particular security notion for secure channels. With the analysis of TLS in mind, Jager et al. [JKSS12] developed the ACCE security notion, which combined the security of key exchange and the subsequent use of the resulting session keys in a secure channel. Their work builds on that of Paterson et al. [PRS11], who introduced extensions of the standard AEAD notions to allow for length hiding; the ACCE approach was subsequently adopted and extended by Krawczyk et al. [KPW13] to analyze a wider set of TLS Handshake Protocol options. Again in the constructive cryptography setting, Badertscher et al. [BMM+15] proposed an abstraction of a secure channel, essentially the constructive-cryptography counterpart of a stateful AEAD scheme, and argued that the security of a modified version of the TLS 1.3 record protocol fulfills the corresponding security goal.

Stream-based channels. Characteristic of all the above-mentioned prior works is that they treat secure channels as providing an *atomic* interface for messages, meaning that the channel is designed only for sending and receiving sequences of discrete messages. However, this only captures a fraction of secure channel designs that are actually used in the real world. In particular, TLS, SSH, and QUIC all provide a *streaming* interface for the applications that use them: applications submit segments (or fragments) of message (or plaintext) streams to an application programming interface (API), and similarly receive fragments of message streams from the API. The sending side may arbitrarily buffer and/or fragment the message stream before encapsulating it for sending.

Moreover, in some cases, even under normal operations, it is not guaranteed by the network that the resulting stream of ciphertext fragments (which we refer to as *ciphertexts* henceforth treating them as opaque bit strings) that is sent will arrive at the receiver with the same pattern of fragmentation, even if the reconstructed message streams are in the end identical.[1] Under adversarial conditions, such guarantees certainly do not hold: for example, TLS runs over TCP and an active man-in-the-middle adversary can tinker with the TCP segments, adding, removing and reordering TLS data at will. Thus practical secure channels need to securely process arbitrarily fragmented ciphertexts. Finally, to make things even more complex, and coming full circle, applications (like HTTP) often rely on stream-oriented secure channels (like TLS) to securely deliver what are actually, in their semantics, atomic messages.

This discussion points to a mismatch between atomic descriptions of secure channels in the cryptography literature and the reality of the operation of secure channels. As one may expect, such mismatches can have negative consequences for security. The starkest example of this comes from the plaintext recovery attack against SSH given by Albrecht et al. [APW09]. Their attack specifically exploits the adversary's ability to deliver arbitrary sequences of SSH packet fragments to the receiver (over TCP) and observe the receiver's behavior in response. The attack is possible despite the analysis of [BKN04] which proved that the SSH secure channel satisfies suitable *atomic* stateful security notions. Related attacks against certain IPsec configurations (and exploiting IPsec's need to handle IP fragmentation) were presented in [DP10]. Attacks highlighting a disjunction between what applications expect and what secure channels provide, in the specific context of HTTP and TLS, can be found in [SP13, BDF$^+$14]. All these attacks highlight deficits of previous approaches to modeling and analyzing secure channels.

Boldyreva et al. [BDPS12] extended the classical, atomic secure-channel confidentiality notion to cover the case of SSH-like stream-based secure channels, broadening the SSH-specific work of Paterson and Watson [PW10]. Subsequently, Albrecht et al. [ADHP16] augmented the model of [BDPS12] with a corresponding notion of integrity. However, while their work allows for fragmented delivery of ciphertexts to the receiver, it still assumes that the encryption process on the sender's side is atomic, meaning that there is a one-to-one correspondence between messages and ciphertexts. This may be the case for SSH when used in interactive sessions, but it is not the case for the tunneling mode of SSH, and never the case for other secure channel protocols. For example, even though the TLS specification [DR08] does not include a formal API definition, it is clear that the design intention is to provide a secure channel for data streams and the application programmer is in practice offered a TCP-like socket interface. As noted above, the sending side can arbitrarily buffer and fragment the message stream when preparing ciphertexts for sending.

Our contributions. In this paper we develop formal functional specifications, security notions, and a construction (using AEAD as a building block) for *stream-based channels*. We also explore how applications, given access to a stream-based channel meeting our security notions, can safely use it to transport

[1]IPsec is a prime example; because of the interaction between IPsec and IP with its fragmentation features, IPsec-protected packets can arrive at their destination in a sequence of fragments, each fragment contained in its own IP packet, and possibly arriving out of order.

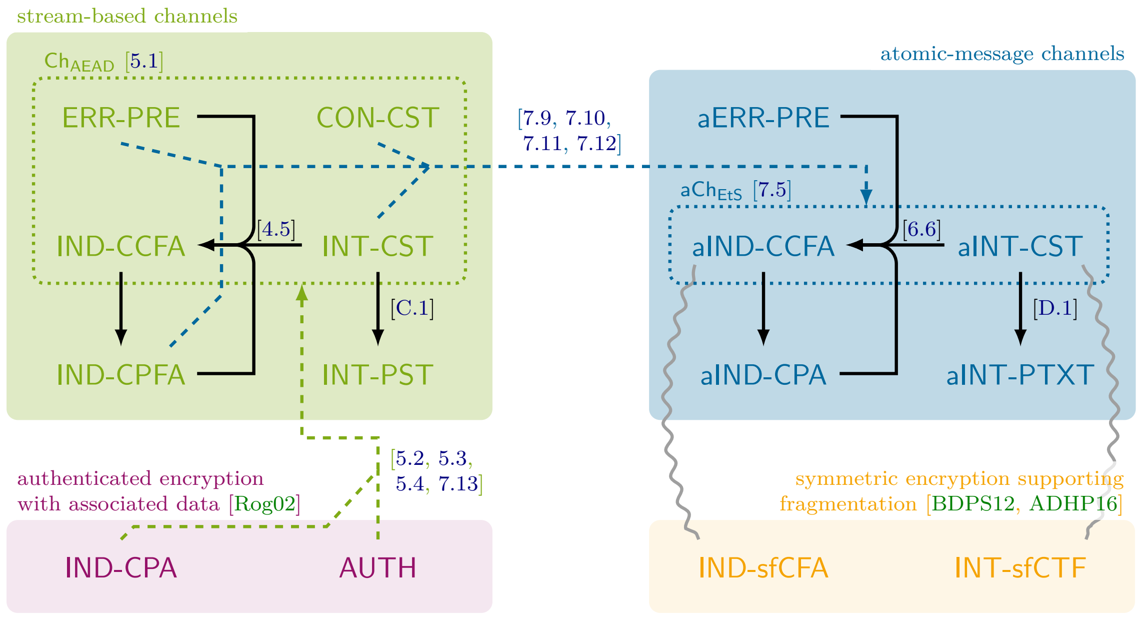

Figure 1: Overview of the notions we establish for stream-based channels and atomic-message channels supporting fragmentation, as well as their formal or conceptual relations to established notions that we demonstrate for authenticated encryption with associated data (AEAD) [Rog02] and symmetric encryption supporting fragmentation [BDPS12, ADHP16]. We provide confidentiality notions in terms of (atomic-message) indistinguishability ((a)IND) under chosen plaintext, plaintext fragment and ciphertext fragment attacks (CPA/CPFA/CCFA), integrity notions in terms of (atomic-message) integrity ((a)INT) of plaintexts, plaintext streams and ciphertext streams (PTXT/PST/CST), and more specific notions for (atomic-message) error predictability ((a)ERR-PRE) and conciseness of ciphertext streams (CON-CST).
Filled rounded rectangular areas indicate the three conceptual notions for channels and the notion of AEAD. Solid arrows indicate implications we establish between security notions within these. Dotted rounded rectangles encompass the security notions achieved by our generic constructions $\mathsf{Ch}_{\mathsf{AEAD}}$ and $\mathsf{aCh}_{\mathsf{EtS}}$. Dashed arrows indicate the security notions required in these constructions from the corresponding building blocks. Wavy lines indicate conceptually analogous security notions between our setting of atomic-message channels supporting fragmentation and the notion of symmetric encryption supporting fragmentation [BDPS12, ADHP16]. Numbers in brackets refer to the corresponding theorems and constructions in our paper. Further discussions can be found in the text.

atomic messages over a fragmenting network via an *atomic-message channel (supporting fragmentation)*.

Our models are in the game-based tradition, and extend those of [BKN04, BDPS12] to handle the streaming nature of the channels that we consider. Figure 1 gives an overview of the notions we establish for both stream-based and atomic-message channels and their relations, including formal and conceptual relations to the established notions of AEAD [Rog02] and symmetric encryption supporting fragmentation [BDPS12, ADHP16].

While our methodology and modeling closely resemble those of [BDPS12], and indeed build upon them, a crucial difference comes in our treatment of the sending (or encrypting) function of a stream-based channel: in [BDPS12], this is still atomic (while decryption is not), whereas in our stream-based channel setting, both the sending and receiving function support streams of data, with potentially arbitrary buffering and fragmentation on the sending and receiving side. This requires careful modification of the confidentiality definitions of [BDPS12]. In addition, we develop suitable integrity notions for the streaming setting whereas [BDPS12] does not consider this aspect. This is important because the (informal) security properties that applications expect a secure channel to provide confidentiality as well as integrity. Concurrent to our work, Albrecht et al. [ADHP16] augmented the model of [BDPS12] with an integrity notion for an analysis of the SSH protocol.

Bringing integrity into the picture for stream-based channels also enables us to prove a composition result analogous to the classical result of [BN00] for symmetric encryption schemes, which states that IND-CPA security in combination with integrity of ciphertexts (INT-CTXT security) guarantees IND-CCA security. This provides an easy route to proving that a given stream-based channel construction provides strong confidentiality (indistinguishability under chosen ciphertext-fragment attacks, or IND-CCFA security) and integrity (integrity of plaintext streams, INT-PST security).

The composition theorem brings an interesting technical challenge to surmount: as was already recognized in [BDPS14] for the classical (atomic) setting, the possibility that realistic models of encryption schemes may involve *multiple* error messages means that the original composition proof of [BN00] does not go through. In [BDPS14], this was overcome by assuming the scheme is such that, from an adversarial point of view, only one of the possible error messages has a non-negligible chance of being produced during operation of the scheme. Here we take a different tack, introducing the concept of *error predictability*, which guarantees the existence of an efficient algorithm that can predict which errors should be output during decryption of a ciphertext stream. We exemplify that such a predictor can exist for a given scheme, even in cases where the analogous conditions to those in [BDPS14] are not satisfied. Our approach can also be used in the atomic-message setting to extend the composition theorem to schemes with distinguishable but predictable errors.

We demonstrate the feasibility of our security notions by providing a generic construction for a stream-based channel that uses AEAD as a component and achieves our strongest confidentiality and integrity notions. The resulting stream-based channel closely mimics the TLS record protocol. That way, our security results provide validation for this important real-world protocol design, whilst fully taking its streaming behavior into account.

Finally, we analyze how applications can safely transport atomic messages over a possibly fragmenting network. To formalize in cryptographic terms the meaning of the latter we propose the notion of *atomic-message channel supporting fragmentation* and develop corresponding confidentiality and integrity notions. With a rigorous security goal in mind we can confirm that the straightforward approach (used, e.g., in HTTP) to encode distinguished end-of-message symbol into the message stream, thus allowing the receiver to reconstruct the message boundaries, does achieve the desired security goal. After looking closely at this approach we develop a generic paradigm, that we call *encode-then-stream*, for building atomic-message channels from stream-based channels, and study its security. The resulting construction provably achieves strong confidentiality and integrity guarantees, as we show, provided that the underlying stream-based channel also offers strong security.

Ultimately, the study of atomic-message security in the presence of ciphertext fragmentation casts a formal light on the truncation [SP13] and 'cookie-cutter' [BDF$^+$14] attacks on HTTP running over TLS, showing how they can be seen as arising from a misunderstanding of the security guarantees that can be provided by a stream-based channel to applications expecting an atomic-message channel. In essence, applications relying on non–integrity-protected end-of-message indicators cannot hope to safely reconstruct atomic messages on the receiving end of a (stream-based) channel.

Further related work. Bhargavan et al. [BFK$^+$13] have developed notions of security for stream-based channels as part of their detailed analysis of the TLS record protocol. Their approach involves expressing channel security properties as types in a programming language, and then formally proving that the type definitions are respected in an adversarial setting (where the adversary is modeled as another program interacting with the code for the send and receive functions of the channel). This can then be related to more traditional game-based security notions (specifically in [BFK$^+$13], to LH-AEAD security as defined in [PRS11]). This is an intriguing approach that deserves further study, as it offers the prospect of obtaining semi-automated proofs of security and can in principle deal with greater complexity than our approach, as well as running code. However, the lack of detail in the definitions of [BFK$^+$13] (for the streaming interface with an atomic LH-AEAD security notion underneath) precludes a detailed comparison with the stream-based security notions that we obtain using a traditional game-based approach. In a subsequent work, Bhargavan et al. [DLFK$^+$17] extended this approach to verify an implementation of TLS 1.3.

A seemingly similar line of work to ours concerns blockwise-adaptive security and on-line symmetric encryption schemes, as developed in [BBKN01, JMV02, FJMV04, FJP04, BT04, Bar07] and recently also taken into account within the CAESAR competition [Ber] and authenticated encryption [FFL12, HRRV15]. There, the schemes operate in an on-the-fly manner, processing one fixed-size block of plaintext or ciphertext at a time; meanwhile the adversary is given access to blockwise encryption (and possibly decryption) oracles. Beyond that, Tsang et al. [TSS09] consider streamwise (on-line) authenticated encryption in the stateless setting (i.e., for a single fragmented ciphertext, treated as a 'stream'), with a focus on latency and overhead on resource-constrained devices. However, in these papers messages and ciphertexts are ultimately regarded as discrete entities, rather than as streams of message and ciphertext fragments as in our treatment. Moreover, the encryption and decryption interfaces presented by an on-line encryption scheme and a stream-based channel (as we define it here) are rather distinct. These differences makes the two lines of work in fact quite incomparable. Of course, our work is trivially distinguished from the vast body of research on *stream ciphers* and on the treatment of specific cryptographic problems in streaming contexts such as [OS05] or the foundational work of [PY14] on the feasibility of cryptography with streaming devices.

Paper organization. After introducing some basic notation and terminology in Section 2, we present in Section 3 our formal definition for stream-based channels. Section 4 contains our security notions for confidentiality and integrity of stream-based channels as well as our composition theorem. In Section 5 we provide a generic construction of stream-based channels from AEAD. Section 6 turns to discussing how to provide a secure atomic-message interface on top of a fragmenting network and which security guarantees such an interface should provide. To this end, we formalize the notion of atomic-message channels supporting ciphertext fragmentation as well as corresponding confidentiality and integrity properties, and study their relations. In Section 7 we propose a generic paradigm, encode-then-stream, for building atomic-message channels from stream-based channels, and assess its security. We conclude with open questions arising from this work in Section 8.

Comparison with proceedings version. A preliminary (proceedings) version of this work appeared as [FGMP15]. The proceedings version only contained the definition, security, and generic construction of stream-based channels (cf. Sections 3–5 of this paper). In this version we provide detailed proofs for the implications among stream-based security notions, the composition theorem, and the security of our generic construction. We also modify the IND-CCFA and INT-CST notions to correct a flaw in the security properties presented in [FGMP15] which also appears in the confidentiality notion of [BDPS12] (the original notions failed to exclude a class of trivial attacks; see Section 4 for the technical details).

In the present work we additionally study how applications can safely use a stream-based channel to transport atomic messages over a possibly fragmenting network. To this end, we formalize in Section 6 the notion of atomic-message channels supporting fragmentation, similar in spirit to [BDPS12], and corresponding security properties. We then provide a generic 'encode-then-stream' construction in Section 7, generalizing the common approach of applications and ensuring provably-secure atomic-message transport over a stream-based channel.

2 Preliminaries

Notation. Let Σ be an alphabet and $s \in \Sigma^*$ be a string, where $s = \varepsilon$ denotes the empty string. By $|s|$ we denote the length of the string s, by $s[i] \in \Sigma$ the i-th character of the string (where $s[1]$ is the first character for a non-empty string), and by $s[i, \ldots, j]$ the substring from and including $s[i]$ up to and including $s[j]$, i.e., $s[i, \ldots, j] = s[i] \parallel \ldots \parallel s[j]$. Given two strings $s, t \in \Sigma^*$ we write $s \preccurlyeq t$ to indicate that s is a *prefix* of t, i.e., there exists $r \in \Sigma^*$ such that $s \parallel r = t$; in this case we write $r = t \,\%\, s$. Similarly, we write $s \prec t$ to indicate that s is a *strict prefix* of t, i.e., $s \preccurlyeq t$ and $s \neq t$. We denote the longest common prefix of s and t by $[s, t] = [t, s]$. Note that $s \preccurlyeq t$ is equivalent to having $[s, t] = s$. With the above notation $s \,\%\, [s, t]$ denotes the suffix of s with the longest common prefix of s and t stripped off.

Let $\mathbf{s} = (s_1, \ldots, s_\ell) \in (\Sigma^*)^\ell$ be a vector of strings for some integer ℓ. (The strings need not be of equal length.) For all $0 \leq i \leq j \leq \ell$ we denote $\mathbf{s}[i] = s_i$ and $\mathbf{s}[i, \ldots, j] = (s_i, \ldots, s_j)$. We say that two vectors $\mathbf{s} = (s_1, \ldots, s_\ell) \in (\Sigma^*)^\ell$ and $\mathbf{t} = (t_1, \ldots, t_{\ell'}) \in (\Sigma^*)^{\ell'}$ are equal, and write $\mathbf{s} = \mathbf{t}$, if and only if $\ell = \ell'$ and for all $i \in \{1, \ldots, \ell\}$ it holds $s[i] = t[i]$. The conversion of a vector into a string is simply performed via the concatenation operation $\parallel\mathbf{s} = s_1 \parallel \ldots \parallel s_\ell$. By convention, the concatenation of an empty vector $()$ is the empty string ε. Slightly overloading notation, we denote the merge of two vectors $\mathbf{s} = (s_1, \ldots, s_\ell)$ and $\mathbf{t} = (t_1, \ldots, t_{\ell'})$ as $\mathbf{s} \parallel \mathbf{t} = (s_1, \ldots, s_\ell, t_1, \ldots, t_{\ell'})$. We also indicate by $\mathbf{s} \preccurlyeq \mathbf{t}$ that $\mathbf{s}$ is a prefix of $\mathbf{t}$, i.e., $\ell \leq \ell'$ and $s_1 = t_1, \ldots, s_\ell = t_\ell$, and in this case we denote by $\mathbf{t} \,\%\, \mathbf{s}$ the (potentially empty) vector $\mathbf{u}$ such that $\mathbf{t} = \mathbf{s} \parallel \mathbf{u}$. We write $\mathbf{s} \prec \mathbf{t}$ to indicate that $\mathbf{s}$ is a strict prefix of $\mathbf{t}$. Similarly, we denote by $[\mathbf{s}, \mathbf{t}]$ the longest vector that is a prefix of both $\mathbf{s}$ and $\mathbf{t}$.

Channel Terminology. Before defining channels formally, it is useful to settle some terminology. Our syntax reflects the generic functionality that a channel should provide, i.e., allowing a sender to transmit messages and a receiver to obtain them in a reliable way. In particular, the notion of a channel is independent of the targeted security properties (like confidentiality or integrity) as these may vary from one specific application to another. While, e.g., a secure channel is generically thought of as being realized via *stateful authenticated encryption*, an authenticated channel might choose to leave confidentiality aside and provide only integrity. We prefer to keep a higher level of abstraction and explicitly separate the generic notion of a channel from its building blocks or targeted security guarantees and hence define sending (Send) and receiving (Recv) algorithms rather than encryption and decryption algorithms.

3 Stream-Based Channels

We capture the functionality of channel protocols that offer a reliable transmission of *streams* like the Transmission Control Protocol (TCP) [Pos81] and, in a second step, we define confidentiality and integrity properties expected from (stream-based) secure channel protocols like the Transport Layer Security (TLS) record protocol [DR08] or the Secure Shell (SSH) Binary Packet Protocol [YL06b].[2] To do so we first need to define the syntax of stream-based channels that, in contrast to previous models for channels, send fragments of a message (or plaintext) *stream* rather than atomic messages. In order to remain close to real-world implementations we restrict both the message space and the ciphertext space to the set of

[2]Our model inherently assumes that, in a benign scenario, ciphertext fragments are delivered reliably and in order (i.e., in a TCP-like manner). While we recognize that efficient and secure transmission protocols can be designed also on top of unreliable protocols like the User Datagram Protocol (UDP) [Pos80] as done, e.g., in Google's Quick UDP Internet Connections (QUIC) protocol [QUI], we deem these approaches orthogonal or unrelated to our work. In such cases, a reliable and ordered stream transmission can be implemented *non-cryptographically* either by TCP-like preprocessing of the UDP datagrams before handing them over to a stream-based channel according to our definition or by postprocessing UDP datagrams which are encrypted and authenticated individually (e.g., using an AEAD scheme).

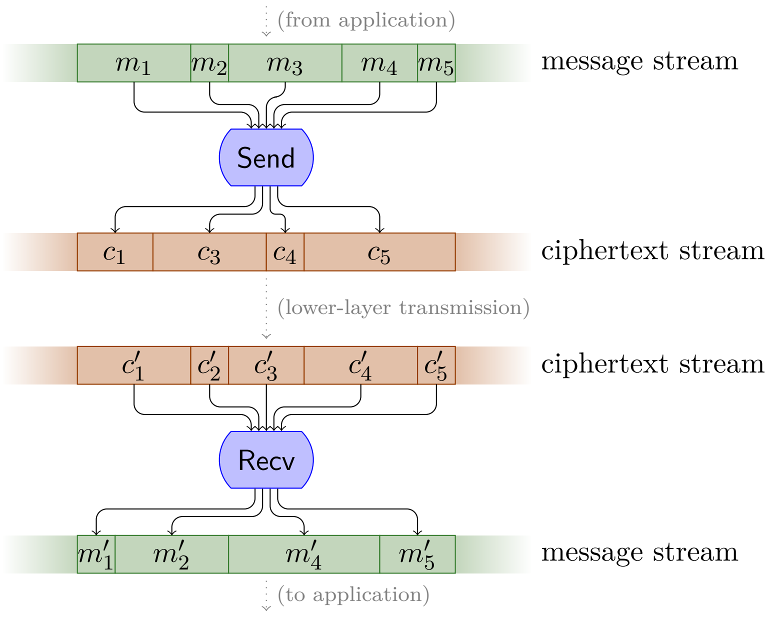

Figure 2: Illustration of the behavior of the Send and Recv algorithms of a stream-based channel, indicating the message and ciphertext fragments being sent (m_i resp. c_i) and received (m'_i resp. c'_i). Note that, due to buffering, output ciphertexts and messages (c_i resp. m'_i) can be empty.

bit strings, where we understand 'messages' and 'ciphertexts' not as atomic units, but as fragments (i.e., substrings) of a message stream and a ciphertext stream.

Additionally, we do not stipulate a particular input/output behavior on the sender side, but instead allow the sending algorithm Send to process input data at its discretion, e.g., implementing some form of buffering before sending. We enforce sending out particular chunks of the message stream by employing the established concept of 'flushing a stream' known from network socket programming, and provide the Send algorithm with an additional *flush* flag $f \in \{0, 1\}$ which, if set to $f = 1$, ensures that all the message fragments fed so far are sent out instantaneously. Jumping ahead, in our security model this choice conservatively also allows the adversary to control the flush flag. If the flush flag is set to zero, Send may internally decide to keep accepting more message fragments or to send out a ciphertext fragment, depending on its implementation and resources.

We remark that our model also captures real-world channels that, instead of offering an explicit flushing mechanism, buffer their input until a specified timeout is reached. In such scenarios, the adversary is simply given control over the timeout through controlling the flush flag.[3]

In our definition below for any message fragment m processed by Send we denote by c the (potentially empty) resulting ciphertext. We stress that c should not be interpreted as an encapsulation of m (i.e., we do not require that c decrypts to m, as we expand later) but just as a label for the output of Send on input m and the current state. In particular, letting Send output empty ciphertexts allows the algorithm to buffer message fragment for later sending. Similar considerations hold for Recv, which may buffer ciphertext fragments before returning a non empty message fragment. Figure 2 illustrates the behavior of the sending and receiving algorithms of a stream-based channel.

We proceed by defining syntax and correctness of stream-based channels.

[3]While any practical channel implementation will send buffered data out eventually (on explicit request or on timeouts), note that our model is general enough to also capture channels that do not offer any control over flushing: for this it suffices to consider a restricted channel interface where the flush flag is fixed to $f = 0$.

Definition 3.1 (Syntax of stream-based channels)**.** *A* stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated sending and receiving state space $\mathcal{S}_S$ resp. $\mathcal{S}_R$ and error space $\mathcal{E}$, where $\mathcal{E} \cap \{0,1\}^* = \emptyset$, consists of three efficient algorithms:*

- Init*. On input a security parameter 1^λ, this probabilistic algorithm outputs initial states* $\mathsf{st}_{S,0} \in \mathcal{S}_S$, $\mathsf{st}_{R,0} \in \mathcal{S}_R$ *for the sender and the receiver, respectively. We write* $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow_{\$} \mathsf{Init}(1^\lambda)$*.*
- Send*. On input a state* $\mathsf{st}_S \in \mathcal{S}_S$*, a message fragment $m \in \{0,1\}^*$, and a flush flag $f \in \{0,1\}$, this (possibly) probabilistic algorithm outputs an updated state* $\mathsf{st}'_S \in \mathcal{S}_S$ *and a ciphertext fragment* $c \in \{0,1\}^*$*. We write* $(\mathsf{st}'_S, c) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_S, m, f)$*.*
- Recv*. On input a state* $\mathsf{st}_R \in \mathcal{S}_R$ *and a ciphertext fragment $c \in \{0,1\}^*$, this deterministic algorithm outputs an updated state* $\mathsf{st}'_R \in \mathcal{S}_R$ *and a message fragment $m \in (\{0,1\} \cup \mathcal{E})^*$. We write* $(\mathsf{st}'_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$*.*

Given a state pair $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$, an integer $\ell \geq 0$, and tuples of message fragments $\mathbf{m} = (m_1, \ldots, m_\ell) \in (\{0,1\}^*)^\ell$ and of flush flags $\mathbf{f} = (f_1, \ldots, f_\ell) \in \{0,1\}^\ell$, let $(\mathsf{st}_S, \mathbf{c}) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_{S,0}, \mathbf{m}, \mathbf{f})$ be shorthand for the sequential execution $(\mathsf{st}_{S,1}, c_1) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_{S,0}, m_1, f_1), \ldots, (\mathsf{st}_{S,\ell}, c_\ell) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_{S,\ell-1}, m_\ell, f_\ell)$ with $\mathbf{c} = (c_1, \ldots, c_\ell)$ and $\mathsf{st}_S = \mathsf{st}_{S,\ell}$. For $\ell = 0$ we define $\mathbf{c}$ to be the empty vector and $\mathsf{st}_{S,\ell} = \mathsf{st}_S$ to be the initial state. We use an analogous notation for the receiver's algorithm.

Correctness of stream-based channels should guarantee that if, after initialization, Send is fed with a message stream, and (a prefix of) the corresponding ciphertext stream is then processed by Recv, then no matter how the ciphertexts are fragmented at the sender's side and re-fragmented at the receiver's side (provided that the order of the bits is preserved), then the returned message stream is a prefix of the initial message stream. Moreover, when Recv consumes a full ciphertext fragment generated by a call to Send with the flush flag set to 1, the stream output by Recv should contain all the message fragments input to Send up to that call.

More precisely, if the receiver obtains (in an arbitrarily fragmented way) a prefix $\| \mathbf{c}'$ of the string of ciphertexts $\| \mathbf{c}$ created by the sender for an input message vector $\mathbf{m} \in (\{0,1\}^*)^\ell$ and flush flag vector $\mathbf{f} \in \{0,1\}^\ell$, and if string $\| \mathbf{c}'$ contains the concatenation $c_1 \| \ldots \| c_i$ of the first i ciphertexts of $\mathbf{c}$, then we require that the message string $\| \mathbf{m}'$ returned on the receiver's side contains as a prefix the concatenation $m_1 \| \ldots \| m_i$ of the first i messages of $\mathbf{m}$ for all indices $i \in \{0\} \cup \{j : f_j = 1\}$ for which the corresponding call to Send flushed its buffer (if all flush flags f_j are set to zero then the above concatenations of ciphertexts and messages are empty and the correctness condition is trivially fulfilled). In particular, this requires that the receiver must output the full and correct message stream if the last Send call has the flush flag f_ℓ set to 1.

Definition 3.2 (Correctness of stream-based channels)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel. We say that* Ch *provides* correctness *if for all state pairs* $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow_{\$} \mathsf{Init}(1^\lambda)$*, all $\ell, \ell' \geq 0$, all choices of the randomness for algorithms* Init, Send *and* Recv*, all message-fragment vectors* $\mathbf{m} \in (\{0,1\}^*)^\ell$*, all flush-flag vectors* $\mathbf{f} \in \{0,1\}^\ell$*, all sending output sequences* $(\mathsf{st}_{S,\ell}, \mathbf{c}) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_{S,0}, \mathbf{m}, \mathbf{f})$*, all ciphertext-fragment vectors* $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$*, and all receiving output sequences* $(\mathsf{st}'_{R,\ell'}, \mathbf{m}') \leftarrow \mathsf{Recv}(\mathsf{st}_{R,0}, \mathbf{c}')$*, we have for any $i \in \{0\} \cup \{j : f_j = 1\}$ that*

$$\| \mathbf{c}[1, \ldots, i] \preccurlyeq \| \mathbf{c}' \preccurlyeq \| \mathbf{c} \implies \| \mathbf{m}[1, \ldots, i] \preccurlyeq \| \mathbf{m}' \preccurlyeq \| \mathbf{m}.$$

Remark 3.3. Note that the receiver's output alphabet consists of bits and of distinct error symbols of the set $\mathcal{E}$; correctness therefore implies that the receiver does not output error symbols for genuine ciphertext streams.

Remark 3.4. It is instructive to compare our correctness definition with that of Boldyreva et al. [BDPS12]. There, correctness requires that if a sequence $\mathbf{m}$ of discrete messages is encrypted, and the resulting ciphertext stream $\|\mathbf{c}$ is then decrypted (possibly in a fragmented manner), then the obtained message sequence (when message separators ¶ are removed) is identical to the original sequence $\mathbf{m}$. In the special case of a single message, this implies that encryption 'always flushes' in the setting of [BDPS12], and is in turn the reason why encryption is necessarily an atomic operation. By contrast, in our setting the Send algorithm is equipped with a flush flag and, when the latter is set to zero, potentially the entire message fragment is buffered for delayed sending. This is, then, an essential difference between the setting of Boldyreva et al. [BDPS12] and the streaming one. An additional difference is that the correctness condition in [BDPS12] is stronger than ours as it incorporates a certain amount of robustness. More specifically, the sequence of ciphertext fragments $\mathbf{c}'$ submitted for decryption in the correctness definition of [BDPS12] may extend the sequence produced by encryption (in other words, $\|\mathbf{c}$ is only required to be a prefix of $\|\mathbf{c}'$ in order for decryption to still work correctly up to $\|\mathbf{c}$). Such cases will be dealt with by our integrity notions.

4 Security for Stream-Based Channels

In this section we introduce confidentiality and integrity notions for stream-based channels and study the relations among these notions. Our notions are game-based and extend security properties for stateful authenticated encryption [BKN04] to the streaming setting. We specify these properties in terms of asymptotic security; analogous notions in the concrete setting are easy to infer.[4]

4.1 Confidentiality

Following the approach of Bellare et al. [BKN04] for stateful encryption we define confidentiality in terms of left-or-right indistinguishability of ciphertexts. We recall that chosen-plaintext attacks (CPA) for stateful encryption are modeled through a game in which $\mathcal{A}$ is given a left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ that, upon being queried on pairs of messages (m_0, m_1), returns encryptions of either the 'left' messages m_0 or the 'right' messages m_1, depending on a secret bit $b \in \{0,1\}$. The game to model chosen-ciphertext attacks (CCA) is similar but additionally provides the adversary with a decryption oracle $\mathcal{O}_{\mathsf{Recv}}$ that $\mathcal{A}$ can query on ciphertexts of its choice, obtaining the corresponding decrypted messages except for *in-sync* ciphertexts (i.e., the sequence of ciphertexts output by the left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ whose decryption would reveal the challenge sequence m_b by correctness). In both games $\mathcal{A}$'s goal is to determine the bit b. We now adapt these notions to the stream-based setting.

As for the case of symmetric encryption supporting ciphertext fragmentation—introduced by Boldyreva et al. in [BDPS12]—our security notions should reflect that the algorithms of a stream-based channel support processing of arbitrary fragments of the message stream and of the ciphertext stream respectively. However, while Boldyreva et al. consider only fragmented decryption (i.e., the encryption process is atomic) and therefore focus their attention on the CCA setting, in our case the fragmentation at the sender also affects the adversarial capabilities in the CPA setting. We hence define two new indistinguishability notions, one for *chosen plaintext-fragment attacks* (IND-CPFA) and one for *chosen ciphertext-fragment attacks* (IND-CCFA). The corresponding experiments, that we denote by $\mathsf{Expt}^{\mathsf{IND\text{-}atk},b}_{\mathsf{Ch},\mathcal{A}}$ for $\mathsf{atk} \in \{\mathsf{CPFA}, \mathsf{CCFA}\}$, are depicted in Figure 3.

Before specifying in detail the logic of our experiments we introduce some useful notation. Within the experiments we denote by C_S the concatenation of the ciphertext fragments sent so far; similarly, we

[4]It is straightforward to define a concrete notion of security by considering the advantage of the adversary as a concrete function of its running time, the numbers of oracle queries, and bounds on the size of the input streams for oracle queries.

write C_R for the concatenation of the ciphertext fragments queried to the receiver. We abbreviate C_S and C_R as the sent (ciphertext) stream and the received (ciphertext) stream, respectively. We say that the sent and received streams are in-sync, or equivalently that they match, if the latter is a prefix of the former ($C_R \preccurlyeq C_S$). If $C_S \not\preccurlyeq C_R$ and $C_R \not\preccurlyeq C_S$ then we say that the two streams deviate, or equivalently that they are out-of-sync. In the experiment we keep a flag sync to indicate whether the above two streams are in-sync ($\mathsf{sync} = 1$) or not. As soon as the streams go out-of-sync we also say that also the oracle $\mathcal{O}_{\mathsf{Recv}}$ goes out-of-sync.

Adapting the CPA experiment to the stream-based settings is relatively easy. We do so by incorporating the flush flag f into the oracle $\mathcal{O}_{\mathsf{LoR}}$ and letting the adversary also specify the value of f; this provides $\mathcal{A}$ with the same interface for Send as that of an application. For each query (m_0, m_1, f) the oracle $\mathcal{O}_{\mathsf{LoR}}$ operates as follows: after checking that the message fragments m_0 and m_1 have the same bit length, it invokes Send on input the current state st_S, message m_b, and flush flag f, it records the resulting ciphertext fragment c into the ciphertext stream C_S, and finally gives c to the adversary.

Formalizing a sound security notion for the CCA setting, where the adversary can also obtain the output of Recv for chosen ciphertext-fragments, turns out to be more delicate. Ideally, we envision a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ that lets $\mathcal{A}$ see as much decrypted plaintext as possible without enabling trivial attacks. Following this principle we mimic the strategy of Bellare et al. [BKN04] to model stateful encryption security by declaring $\mathcal{O}_{\mathsf{Recv}}$ to be in-sync—thus instructing it to artificially suppress the output of Recv—as long as the adversary supplies a prefix of the original ciphertext stream output by the left-or-right oracle.

There is, however, a definitional challenge to surmount. In contrast to stateful encryption where messages input to the encryption algorithm are considered as atomic units and correspond in a one-to-one manner to the output ciphertexts, in the streaming scenario messages and ciphertexts can be arbitrarily fragmented. Therefore, it is not clear a priori how to translate deviations of the ciphertext sequences into deviation of the message sequences. To adapt the suppression mechanism of Bellare et al. to the streaming setting we need to determine at which point exactly the ciphertext stream at the receiver should be considered out-of-sync.

Synchronization/suppression mechanism for symmetric encryption supporting fragmentation. The experiment for indistinguishability against chosen-fragment attacks ($\mathsf{IND\text{-}sfCFA}$) proposed by Boldyreva et al. [BDPS12, Definition 4] in the context of symmetric encryption supporting ciphertext fragmentation stays close to the original definitions of [BKN04] and conservatively defines synchronization to be lost *at the ciphertext boundaries*. That is, $\mathcal{O}_{\mathsf{Recv}}$ suppresses the output of Recv up to the longest sequence of genuine ciphertexts (recall that in [BDPS12] the encryption algorithm is atomic, and hence it outputs entire ciphertexts rather than fragments, making it possible to identity ciphertext boundaries in a security experiment). In a run of the $\mathsf{IND\text{-}sfCFA}$ experiment let $m_1, \ldots, m_i$ be the messages processed by Send and let $c_1, \ldots, c_i$ be the resulting ciphertexts, let $j \leq i$ be maximal such that the receiving algorithm processes entirely the sequence of the first j sent ciphertexts, and suppose that the stream C_R of ciphertext fragments processed by Recv deviates from the genuine sequence $C_S = c_1 \parallel \cdots \parallel c_i$. Then synchronization is considered to be lost from the first bit following ciphertext c_j, and hence $\mathcal{O}_{\mathsf{Recv}}$ suppresses exactly the first j sent messages $m_1, \ldots, m_j$ from the output of Recv.

As we show in the following examples, this option is inappropriate in a stream-based setting where the ciphertext fragments output by Send do not necessarily correspond one-to-one to the input message-fragments.

Consider the case of TLS and the Send algorithm being called on a $(2^{14} + 1)$-byte input message with the flush flag set to 1—mimicking the behavior of many TLS implementations that keep no send buffer. Obeying the limit of at most 2^{14} bytes of payload in a single TLS record, Send is forced to output a ciphertext fragment containing (at least) two TLS records. According to the IND-sfCFA experiment in [BDPS12], an adversary could forward this fragment to the decryption oracle with the second record modified but the first record untouched. The adversary would then obtain the decryption of *both* records as the IND-sfCFA experiment considers them as jointly forming a *single* ciphertext, hence revealing parts of the challenge message string. Thus, the IND-sfCFA notion would be unachievable for TLS.

As another example assume that a stream-based channel, aiming at confidentiality only, generates the ciphertext stream C_S as the bitwise XOR of the message stream M_S and the output of a stream cipher. In this setting, the IND-sfCFA notion would consider the ciphertext fragments output by Send on the input message fragments to be units that should be either completely kept confidential or, if modified, can be fully leaked to the adversary. However, with the channel exhibiting no further structure, enforcing any block boundaries clearly becomes artificial.

Synchronization/suppression mechanism for stream-based channels. Taking into account that in the streaming scenario the output of Send is a bit stream without any further structure in general, we declare synchronization to be lost starting from the first bit of the receiving ciphertext stream C_R deviating from the genuine stream C_S.

Concretely, suppose that during an execution of the IND-CCFA experiment the adversary causes C_R to deviate from C_S by submitting for decryption a *strict* prefix of the genuine stream followed by additional bits that deviate from C_S (using a more compact notation: $[C_S, C_R] \prec C_S$ and $[C_S, C_R] \prec C_R$). In this case the streams C_S and C_R are considered to be in-sync up to their longest common prefix $[C_S, C_R]$ while the deviating portion $C_R \,\%\, [C_S, C_R]$ is out-of-sync. Given this, we let $\mathcal{O}_{\mathsf{Recv}}$ process ciphertext fragments submitted for decryption by updating C_R with the newly queried fragment and, as long as C_R is a prefix of C_S, invoking Recv (thereby updating the state st_R) on this fragment and suppressing the corresponding output. If the adversary instead submits a fragment that causes C_R to deviate from C_S the oracle stops suppressing and, from then on, the output of Recv is given to the adversary.

Processing the first ciphertext fragment that causes a deviation requires some care, though. The challenging situation is as follows: all fragments submitted for decryption so far are in-sync ($C_R \preccurlyeq C_S$) and the 'next' ciphertext fragment c induces a deviation ($C_R \parallel c \not\preccurlyeq C_S$). Now, if c contains a non-empty in-sync prefix its decryption may trivially reveal challenge bits that should instead be suppressed (as in the above example concerning TLS). To resolve this issue we let the receiving oracle invoke Recv on both the entire fragment c and its longest in-sync prefix $\widetilde{c}$ (the latter is the longest prefix of c that matches C_S), making sure that Recv takes as input the same state st_R for both operations. Let m and $\widetilde{m}$ be the resulting message fragments respectively, and note that $\widetilde{m}$ matches the genuine message stream by correctness. Then $\mathcal{O}_{\mathsf{Recv}}$ suppresses from m its longest prefix that matches the genuine stream (i.e., it suppresses the potentially empty fragment $[m, \widetilde{m}]$) and gives the resulting string $m' = m \,\%\, [m, \widetilde{m}]$ to the adversary. The idea behind this strategy is that any potential challenge bit originating from the in-sync part of c remains hidden from the adversary.

Another subtlety arising from the ciphertext fragmentation concerns the possibility that C_R and C_S are neither in-sync nor out-of-sync. This may happen if the adversary submits to $\mathcal{O}_{\mathsf{Recv}}$ the *entire* (potentially empty) genuine stream of ciphertexts followed by additional bits, making C_R exceed C_S. In such a case the stream C_R does not explicitly deviate from, but only extends, the genuine stream C_S, that is, $C_S \prec C_R$.

One could argue that submitting for decryption any bit that has not been honestly produced at the sender should be considered an active attack and, thus, any part of C_R exceeding C_S should be declared out-of-sync.[5] This intuition turns out to be wrong, though. In fact, the ability to guess a small prefix of the 'next' ciphertext fragment output by Send should not be considered as giving a significant advantage to

[5]This was indeed the argument adopted in [BDPS12] as well as in the proceedings version of this work [FGMP15], and later identified as flawed by Degabriele [Deg16] in May 2016.

```
$\mathsf{Expt}^{\mathsf{IND\text{-}atk},b}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:
1  $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
2  $\mathsf{sync} \leftarrow 1$
3  $C_S \leftarrow \varepsilon$, $C_R \leftarrow \varepsilon$
4  $b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
5  return $b'$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{LoR}}(m_0, m_1, f)$:
6  if $|m_0| \neq |m_1|$ then
7     return $\varepsilon$ to $\mathcal{A}$
8  $(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, m_b, f)$
9  $C_S \leftarrow C_S \parallel c$
10 return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
11 if $\mathsf{sync} = 0$ then // already out-of-sync
12    $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
13    return $m$ to $\mathcal{A}$
14 else if $C_R \parallel c \preccurlyeq C_S$ then // still in-sync
15    $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
16    $C_R \leftarrow C_R \parallel c$
17    return $\varepsilon$ to $\mathcal{A}$
18 else
19    if $C_R \prec [C_R \parallel c, C_S]$ then
          // $c$ deviates or exceeds, contains genuine part
20       $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
21       $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
22       $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
23       $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
24       $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
25    else // $c$ deviates or exceeds, contains no genuine part
26       $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
27    if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
          // deviation, or exceeding portion produces output
28       $\mathsf{sync} \leftarrow 0$
29    $C_R \leftarrow C_R \parallel c$
30    return $m'$ to $\mathcal{A}$
```

Figure 3: Security experiment for *confidentiality* (IND-atk with atk $\in$ {CPFA, CCFA}) of stream-based channels. A CPFA-attacker only has access to the oracle $\mathcal{O}_{\mathsf{LoR}}$.

the adversary. Then, declaring synchronization to be lost with the first bit of C_R exceeding C_S would allow for trivial attacks like the following. The adversary starts by making a guess on the first bit $d' \in \{0,1\}$ output by Send and querying $\mathcal{O}_{\mathsf{Recv}}$ on input fragment d'. As $C_R = d'$ deviates from $C_S = \varepsilon$ this first query desynchronizes the receiving oracle. The adversary proceeds by posing a left-or-right query $(m_0, m_1, 1)$ with $m_0 \neq m_1$, obtains a challenge ciphertext-fragment c', and if its guess was correct (i.e., $d' \preccurlyeq c'$) it submits for decryption the ciphertext fragment $c' \,\%\, d'$, otherwise it terminates. If the adversary correctly guessed the first bit of c' then by correctness it gets from $\mathcal{O}_{\mathsf{Recv}}$ the message $m' = m_b$. This strategy succeeds with probability $\frac{1}{2}$. Merely guessing a small prefix of the next genuine ciphertext fragment which does not produce any output is possible for any stream-based channel, but should not be considered as a success of the adversary.

To exclude the class of attacks like the one just described we adopt the following strategy: we only declare synchronization to be lost when the exceeding portion $C_R \,\%\, C_S$ produces a *non-empty* output—in which case we also conservatively provide the adversary with this output. In other words, we only give credit to the adversary if it is able to predict a non-trivial ciphertext fragment that actually leads to a non-empty plaintext. This allows $\mathcal{O}_{\mathsf{Recv}}$ to later suppress further message bits in case, as a consequence of the adversary posing more left-or-right queries, C_S and C_R end up being matching again (in the sense that $C_R \preccurlyeq C_S$).

To facilitate understanding we illustrate in Figure 4 how $\mathcal{O}_{\mathsf{Recv}}$ processes the first deviating ciphertext fragment ($C_R \parallel c \not\preccurlyeq C_S$) or an exceeding ciphertext fragment ($C_S \prec C_R \parallel c$) by performing two calls to Recv, one on input c and the other on input $\widetilde{c}$.

Putting everything together, we can now unpack from Figure 3 the instructions executed by the $\mathcal{O}_{\mathsf{Recv}}$ oracle upon being queried on a ciphertext fragment c. Its logic treats the two simplest cases first. In case synchronization has been already lost (indicated by the flag sync being 0, line 11) the oracle responds

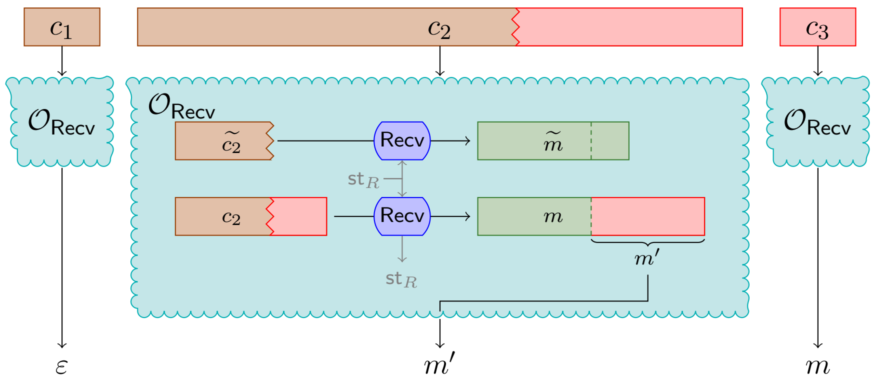

Figure 4: Illustration of the $\mathcal{O}_{\mathsf{Recv}}$ oracle behavior in the $\mathsf{IND\text{-}CCFA}$ experiment from Figure 3 for ciphertext fragment inputs c_1, c_2, c_3 specified by the adversary $\mathcal{A}$ where c_2 deviates from or exceeds the genuine ciphertext stream after the zigzag line. The oracle $\mathcal{O}_{\mathsf{Recv}}$ goes out-of-sync on deviation or if exceeding ciphertext input produces non-empty output.

with the entire output of Recv on input the fragment c. If c is in-sync ($C_R \parallel c \preccurlyeq C_S$, line 14) the oracle invokes Recv on input c but fully suppresses its output. Next consider the case in which c makes C_R extend C_S ($C_R \parallel c \not\preccurlyeq C_S$ but $C_S \preccurlyeq C_R \parallel c$) or causes a deviation ($C_R \parallel c \not\preccurlyeq C_S$ and $C_S \not\preccurlyeq C_R \parallel c$). Here the oracle $\mathcal{O}_{\mathsf{Recv}}$ first checks whether c contains a non-empty, in-sync prefix $\widetilde{c}$ matching C_S which may contain challenge-message bits that should be suppressed. Note that the corresponding check in line 19 evaluates to true if either $C_R \parallel c$ deviates from C_S but c contains a non-empty in-sync prefix $\widetilde{c}$ (thus $[C_R \parallel c, C_S] = C_R \parallel \widetilde{c} \prec C_S$ with $\widetilde{c} \neq \varepsilon$), or $C_R \parallel c$ extends C_S but c contains a non-empty prefix matching C_S (i.e., $[C_R \parallel c, C_S] = C_R \parallel \widetilde{c} = C_S$ also with $\widetilde{c} \neq \varepsilon$). If from the above check it turns out that c contains a non-empty in-sync prefix, the receiving operation is handled by performing a double invocation of Recv on input identical states $\mathsf{st}_R = \widetilde{\mathsf{st}_R}$ and ciphertext fragments c and $\widetilde{c}$ respectively, which yields message fragments m and $\widetilde{m}$, as described earlier. We stress that the second invocation should be considered as an auxiliary step performed by the oracle to establish which portion of m, if at all, shall be suppressed (line 24). Indeed, the latter step is skipped in case c is fully deviating from or fully exceeding C_S (line 25). In either case, the oracle proceeds with determining whether synchronization is lost upon the receipt of c because of a deviation ($C_S \not\preccurlyeq C_R \parallel c$) or because the adversary managed to guess a non-trivial ciphertext fragment leading to a non-emtpy output ($C_S \preccurlyeq C_R \parallel c$ but $m' \neq \varepsilon$). The message fragment m' here denotes the output of Recv corresponding to the deviating or exceeding part of c; this is the actual message fragment that $\mathcal{O}_{\mathsf{Recv}}$ returns to the adversary.

We are now ready to give the formalism of our confidentiality experiments.

Definition 4.1 ($\mathsf{IND\text{-}CPFA}$ and $\mathsf{IND\text{-}CCFA}$ Security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{IND\text{-}atk},b}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* b *be defined as in Figure 3, where* atk *is a placeholder for either* CPFA *or* CCFA*. Within the experiment the adversary* $\mathcal{A}$ *is given access to a (stateful) left-or-right sending oracle* $\mathcal{O}_{\mathsf{LoR}}$ *and, in the case of* $\mathsf{IND\text{-}CCFA}$ *security, a (stateful) receiving oracle* $\mathcal{O}_{\mathsf{Recv}}$*. We say that* Ch *provides* indistinguishability under chosen plaintext-fragment attacks*, respectively* chosen ciphertext-fragment attacks *(*$\mathsf{IND\text{-}CPFA}$*, resp.* $\mathsf{IND\text{-}CCFA}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}atk},1}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}atk},0}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] \right|.$$

Length hiding. Note that we do not consider the extended length-hiding setting that was introduced in [PRS11] to model TLS's variable length padding capability and subsequently incorporated into the ACCE security definition for secure channels in [JKSS12, KPW13]. While our work could conceivably be extended to incorporate length hiding, it remains unclear to us what its value would be in the setting of *streaming* channels, since length hiding is a notion intrinsically connected to atomic messages.

Alternative option to define confidentiality. For the sake of completeness we comment on an alternative, intuitively appealing definition for the receiving oracle in Appendix A. The idea essentially is to split the first deviating or any exceeding fragment c into two parts, its longest in-sync prefix $\widetilde{c}$ and the remaining portion $c \,\%\, \widetilde{c}$, then process both parts through Recv and give only the second output to the adversary. While this option requires no auxiliary invocation of Recv, it results in a strictly weaker confidentiality notion.

Revision of the proceeding version. We point out that the notion of confidentiality against chosen ciphertext-fragment attacks ($\mathsf{IND\text{-}CCFA}$) presented here is a revised version of the notion that appeared in the proceedings version of this work [FGMP15]. As noted earlier, the notion in [FGMP15] failed to exclude trivial attacks where $\mathcal{A}$ merely guesses exceeding bits, a case where synchronization should not be considered lost if no output is produced. Beyond bookkeeping, line 27 contains the core difference between the proceedings version and the present version of the experiment. The check identifies synchronization to be lost if either C_R deviates from C_S or if the exceeding portion leads to a non-empty message. This issue was pointed out to us by Degabriele in May 2016 [Deg16]. The same issue arises in the setting of [BDPS12] and was reported for that setting in [ADHP16]. An analogous revision was applied to the integrity notion for streams, presented in the following section.

On the scope of our confidentiality definition. When formalizing a security goal it is common to develop a notion that is as strong as possible, yet achievable. The inability to foresee new attacks is the main reason for aiming at the strongest notion. However, sometimes this conservative approach leads to security notions that are 'too strong' for some applications, meaning that some schemes which have no actual vulnerability are declared insecure within the model. An example of a security notion that—one might argue—is too strong is $\mathsf{IND\text{-}CCA}$ security for symmetric encryption. For instance, if one starts with an $\mathsf{IND\text{-}CCA}$-secure encryption scheme and modifies it by letting the encryption routine append a redundant bit to each ciphertext and letting the decryption routine ignore that last bit of each ciphertext, the resulting scheme is no longer $\mathsf{IND\text{-}CCA}$-secure. However, adding a redundant bit that is then ignored for decryption should not harm the scheme's confidentiality.[6]

Our $\mathsf{IND\text{-}CCFA}$ notion for stream-based channel may likewise be too strong for some applications. In Appendix B we describe a scheme due to Poettering [Poe16] that, despite being intuitively confidential, is vulnerable to an $\mathsf{IND\text{-}CCFA}$ attack. Briefly, this stream-based channel processes message fragments to be sent by AEAD-encrypting fixed-length chunks of each input fragment, and similarly processes ciphertext fragments to be received by AEAD-decrypting corresponding ciphertext blocks in such a way that, if any of the AEAD decryptions fails, then a distinguished constant string is returned rather than an error. Clearly, the scheme does not provide integrity protection, because the receiver would not be able to detect that the message output stems from an error. However, because of the AEAD security, the message fragment output by Recv on input a deviating ciphertext fragment should only reveal the distinguished string independently of the challenge-message fragment and, thus, confidentiality should not be compromised. The way message output is suppressed in our $\mathsf{IND\text{-}CCFA}$ experiment (cf. Figure 3) however enables an $\mathsf{IND\text{-}CCFA}$ attack on this scheme which is always successful, as described in Appendix B. Exploring possible relaxations of the stream-based $\mathsf{IND\text{-}CCFA}$ confidentiality notion which still uphold an intuitive, strong confidentiality guarantee is an interesting direction for future work.

[6] An alternative notion of confidentiality that precisely aims at resolving this issue was proposed by Canetti et al. [CKN03] as RCCA security (for 'replayable' CCA) in the public-key setting (and can be easily extended to the secret-key setting).

4.2 Integrity

Next, we formalize integrity notions for stream-based channels. We highlight that, while integrity properties for atomic messages (and atomic ciphertexts) are well-understood, no previous work considered integrity in the non-atomic setting. In particular Boldyreva et al. [BDPS12] only addressed confidentiality in the presence of ciphertext fragmentation; their notions were later and concurrently to this work augmented with integrity notions by Albrecht et al. [ADHP16]. We define integrity notions for stream-based channels as refinements of standard (stateful) properties of plaintext integrity ($\mathsf{INT\text{-}sfPTXT}$), resp., ciphertext integrity ($\mathsf{INT\text{-}sfCTXT}$) from [BKN04] and refer to the new properties as *plaintext-stream integrity*, resp., *ciphertext-stream integrity* ($\mathsf{INT\text{-}PST}$, resp., $\mathsf{INT\text{-}CST}$).

Similarly to the setting with atomic messages, $\mathsf{INT\text{-}PST}$ ensures that no adversarial query to the receiving oracle causes the message stream output by Recv to deviate from the message stream input to Send. This notion is quite simple to formulate. Formalizing the stronger $\mathsf{INT\text{-}CST}$ property demands more care. Intuitively, from ciphertext integrity we expect that when processing any 'out-of-sync' ciphertext, the algorithm Recv should return an error message. However, when considering a stream-based interface it may happen that Recv processes an out-of-sync ciphertext which does not yet contain 'enough information' to be recognized as being invalid; in this case the receiving algorithm would buffer (part of) the ciphertext and wait for further fragments until a sufficiently long ciphertext string is available to be processed and deemed as valid or invalid. In such a scenario, a naive adaptation of the $\mathsf{INT\text{-}sfCTXT}$ definition of [BKN04] would allow trivial attacks by declaring successful any adversary that makes the Recv buffer (part of) an out-of-sync ciphertext, without producing non-trivial output. Our notion of ciphertext-stream integrity carefully identifies the case just described and, by letting the receiving oracle wait for further ciphertext fragments, declares the adversary successful only if Recv outputs a non-empty message fragment resulting from an out-of-sync or exceeding portion of the ciphertext stream.

We formalize integrity of plaintext and ciphertext streams through the security experiment $\mathsf{Expt}^{\mathsf{INT\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}$ depicted in Figure 5. The experiment provides the adversary with oracles $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$, where the former grants $\mathcal{A}$ access to algorithm Send under arbitrarily chosen message fragments and the latter gives $\mathcal{A}$ an interface with algorithm Recv. We highlight that, while the sending oracle $\mathcal{O}_{\mathsf{Send}}$ is common for both experiments $\mathsf{INT\text{-}PST}$ and $\mathsf{INT\text{-}CST}$, the receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ follows different procedures in the two cases, as we further explain below.

In the execution of the $\mathsf{INT\text{-}PST}$ experiment, $\mathcal{O}_{\mathsf{Send}}$ maintains in string M_S the stream of all sent message fragments and, analogously, $\mathcal{O}_{\mathsf{Recv}}$ maintains in M_R the stream of all received message fragments (and/or error symbols). The adversary wins the game if it causes M_S and M_R to deviate in such a way that their difference contains more than error symbols. Formally, we demand that the string M_R output by the receiver is not a prefix of the sender's string M_S, but such that this prefix-freeness is not only due to error symbols from $\mathcal{E}$.

In the $\mathsf{INT\text{-}CST}$ experiment oracles $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$ maintain strings C_S and C_R to record the streams of sent ciphertexts, resp. received ciphertext fragments. Furthermore, $\mathcal{O}_{\mathsf{Recv}}$ decides when the adversary wins by inspecting sent and received *ciphertext* streams, an inherently more complex task than looking for deviations in the underlying sequences of sent/received message fragments. Indeed, in a stream-based channel the algorithm Recv may need to buffer several ciphertexts before being able to recover the underlying message stream or detecting that an error occurred; such a behavior is reflected in our experiment. When processing in-sync ciphertexts $\mathcal{O}_{\mathsf{Recv}}$ simply appends each new fragment to C_R. At the moment when an out-of-sync ciphertext fragment or one that exceeds the sent ciphertext stream arrives, the oracle compares the outputs of algorithm Recv when processing (i) the current input ciphertext c and (ii) its longest in-sync prefix $\widetilde{c}$. The adversary wins if $\mathcal{O}_{\mathsf{Recv}}$ outputs more in case (i) than it would in case (ii) and if the difference between the two outputs is a non-empty, valid message. It also wins if it is able to make Recv output a non-empty, valid message with a subsequent out-of-sync ciphertext. As for

```
$\mathsf{Expt}^{\mathsf{INT\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:
$(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
$\mathsf{sync} \leftarrow 1$
$\mathsf{win} \leftarrow 0$
$M_S, C_S \leftarrow \varepsilon,\ M_R, C_R \leftarrow \varepsilon$
$\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
return $\mathsf{win}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m, f)$:
$(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, m, f)$
$M_S \leftarrow M_S \parallel m$
$C_S \leftarrow C_S \parallel c$
return $c$ to $\mathcal{A}$

INT-PST
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
$(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
$M_R \leftarrow M_R \parallel m$
if $M_R \,\%\, [M_R, M_S] \notin \mathcal{E}^*$ then
    $\mathsf{win} \leftarrow 1$
return $m$ to $\mathcal{A}$
```

```
INT-CST
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
if $\mathsf{sync} = 0$ then // already out-of-sync
    $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
    if $m \notin \mathcal{E}^*$ then $\mathsf{win} \leftarrow 1$
else if $C_R \parallel c \preccurlyeq C_S$ then // still in-sync
    $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
    $C_R \leftarrow C_R \parallel c$
else
    if $C_R \prec [C_R \parallel c, C_S]$ then
        // c deviates or exceeds, contains genuine part
        $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
        $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
        $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
        $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
        $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
    else // c deviates or exceeds, contains no genuine part
        $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
        $m \leftarrow m'$
    if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
        // deviation, or exceeding portion produces output
        $\mathsf{sync} \leftarrow 0$
    $C_R \leftarrow C_R \parallel c$
    if $m' \notin \mathcal{E}^*$ then $\mathsf{win} \leftarrow 1$
return $m$ to $\mathcal{A}$
```

Figure 5: Security experiment for *integrity* (INT-atk with atk $\in$ {PST, CST}) of stream-based channels. A PST-attacker is provided with access to the left $\mathcal{O}_{\mathsf{Recv}}$ oracle (INT-PST), whereas a CST-attacker is instead granted access to the oracle on the right-hand side (INT-CST).

confidentiality (see the discussion in Section 4.1) we consider the $\mathcal{O}_{\mathsf{Recv}}$ oracle to go out of sync (and set $\mathsf{sync} \leftarrow 0$) if the ciphertext fragment deviates from the corresponding bits in the sent ciphertext stream or, when it merely exceeds the sent stream, if the output of Recv for the exceeding part is non-empty.[7]

Definition 4.2 (INT-PST and INT-CST Security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{INT\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 3, where* atk *is a placeholder for either* PST *or* CST*. Within the experiment, the adversary* $\mathcal{A}$ *is given access to a sending oracle* $\mathcal{O}_{\mathsf{Send}}$ *and a receiving oracle* $\mathcal{O}_{\mathsf{Recv}}$*. We say that* Ch *provides* integrity of plaintext streams*, respectively* ciphertext streams *(*INT-PST*, resp.* INT-CST*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{INT\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{INT\text{-}atk}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right].$$

Remark 4.3. Our definitions of integrity do not preclude from deeming those channels secure in which message bits can be output as a result of the adversary delivering *partial* ciphertexts to the Recv oracle. This is because in the streaming setting we care about the adversary's ability to force the receiver to accept message fragments corresponding to a part of the ciphertext stream that has gone out-of-sync, without

[7]As already discussed in Section 4.1, withholding synchronization loss for exceeding fragments that produce no output eliminates the trivial attack which was present in the proceedings version of this paper [FGMP15]. There, the adversary could guess (and input to $\mathcal{O}_{\mathsf{Recv}}$) the first bit(s) of the next $\mathcal{O}_{\mathsf{Send}}$ output and, if successful, feed in the remaining ciphertext. With $\mathcal{O}_{\mathsf{Recv}}$ considered out-of-sync on these first bit(s), although generating no output, this attack illegitimately was considered a successful break of ciphertext-stream integrity.

attaching importance to ciphertext boundaries. Hence, this is quite distinct from the usual atomic setting. Of course, applications making use of a streaming channel may wish to recover a secure channel for atomic messages, in a more traditional sense. For example, this is the case for HTTP running over TLS and, as noted in the introduction, has been a source of confusion for developers and led to concrete attacks on protocols such as TLS [SP13, BDF$^+$14]. We will examine this situation in greater detail in Section 6.

We further note that stream-based integrity providing weaker guarantees than atomic-message integrity seems to be an intrinsic consequence of the nature of stream-based channels. In particular, apparent avenues for strengthening the given integrity definition lead to notions which are clearly inappropriate in the streaming setting. On the one hand, requiring a channel to output an error immediately after processing the first bit deviating from the sent ciphertext stream is, for most constructions, an unattainable goal as it is in general impossible to decide if an initial bit received is genuine or not.[8] On the other hand, requiring that a channel does not output any message bit until a full ciphertext output by Send is received inappropriately enforces an atomic structure on the channel, i.e., basically the one of [BDPS12]. The latter notion, as already discussed, is too strong for channels that, like TLS, might output ciphertexts which contain multiple, independent parts.

4.3 Relations Amongst Notions and Generic Composition Theorem

We now explore relations between confidentiality and integrity—well-established for atomic messages by [BN00, BKN04] and follow-up work, culminating in [NRS14]—and investigate whether these relations can be lifted to our streaming setting. We highlight that, since integrity for encryption schemes supporting ciphertext fragmentation was not addressed in [BDPS12], we are the first to consider such relations in the presence of fragmentation.

Ideally we would like to translate the classic implications $\mathsf{IND\text{-}CCA} \implies \mathsf{IND\text{-}CPA}$, $\mathsf{INT\text{-}CTXT} \implies \mathsf{INT\text{-}PTXT}$, and the powerful composition result $\mathsf{IND\text{-}CPA} \wedge \mathsf{INT\text{-}CTXT} \implies \mathsf{IND\text{-}CCA}$, all from [BN00], to the realm of stream-based channels. It is immediate to see that, as in the setting where messages are atomic, the stronger notions implies the weaker ones for both confidentiality and integrity individually. Unfortunately, when integrity and confidentiality are targeted simultaneously, the situation for streams is fundamentally more challenging.

Recall that, in the atomic-message setting, the proof of the composition theorem in [BN00] proceeds in two steps: starting from the $\mathsf{IND\text{-}CCA}$ game, one first bounds the probability that the adversary submits a valid decryption query distinct from an output of the encryption oracle by using the $\mathsf{INT\text{-}CTXT}$ advantage. This then allows a reduction to the $\mathsf{IND\text{-}CPA}$ experiment (now assuming integrity of ciphertexts), simply by answering all decryption queries with the distinguished error symbol $\perp$. As already noted by Boldyreva et al. [BDPS14], the same proof strategy does not work for schemes that have multiple decryption error symbols (which models common real-world behavior of encryption schemes). This is because the reduction can no longer (in general) know which one of the several possible error symbols should be output when simulating decryption.

Thus the classic result $\mathsf{IND\text{-}CPA} \wedge \mathsf{INT\text{-}CTXT} \implies \mathsf{IND\text{-}CCA}$ already does not follow in the situation where multiple error messages are possible, not even considering streaming. Worse, [BDPS14] shows that, in the multiple decryption error setting, there exist schemes that are secure in both $\mathsf{IND\text{-}CPA}$ and $\mathsf{INT\text{-}CTXT}$ senses, yet are not $\mathsf{IND\text{-}CCA}$ secure. We show later in this section that similar issues arise for stream-based channels, even when restricting to the case of single error messages. Specifically, fragmentation at the receiver's side makes it harder to emulate a receiving oracle for the $\mathsf{IND\text{-}CCFA}$ experiment given a receiving oracle for the $\mathsf{INT\text{-}CST}$ game.

[8] Indeed, stream-based integrity does not enforce that Recv outputs an error on deviating ciphertext fragments at all, but is also satisfied by a channel providing no output (i.e., the empty string) in such cases.

As a remedy we propose an adapted version of the composition theorem, resurrecting the result both in our streaming setting and in the case of multiple errors that was considered in [BDPS14]. However this result can be proven only at the cost of introducing further assumptions on the output behavior of the receiving algorithm. The conditions for the composition theorem may initially look quite demanding but, as we confirm in Section 5, there exist natural schemes that satisfy the required conditions. Moreover, the use of the composition theorem is not the only route to achieving IND-CCFA security: for specific schemes it may be possible to prove IND-CCFA security directly.

Confidentiality. A study of the experiments in Figure 3 immediately shows that IND-CCFA security implies IND-CPFA security, since an attacker in the IND-CPFA game only needs to emulate the left-or-right oracle to provide a faithful simulation of the IND-CCFA game, and can trivially do so by relaying all encryption queries to its own left-or-right oracle.

Integrity. Assume that ciphertext-stream integrity (INT-CST) from Definition 4.2 holds for a stream-based channel. Then the channel also provides integrity of plaintext streams (INT-PST) and the security reduction is tight. To see why consider the integrity experiment depicted in Figure 5: given the INT-CST property, every efficient adversary either never produces a ciphertext stream C_R that deviates from the ciphertext stream C_S (hence, by correctness, no deviation will occur in the underlying message streams) or, if it generates a stream C_R that does deviate from C_S, by INT-CST the underlying message streams will only differ by an error string. We give a formal proof in Appendix C.

Generic composition. As explained earlier, standard arguments to prove the composition theorem do not apply in the streaming setting. The issue here is that losing the integrity game does not make the output of $\mathcal{O}_{\mathsf{Recv}}$ (in the confidentiality game) predictable. Therefore, any strategy which allows the recovery of the composition theorem should make it possible to forecast the output behavior of the receiving algorithm when certain conditions are met. In line with this observation we introduce a new notion, so-called *error predictability*, which precisely formalizes the ability to efficiently predict (part of) the output of Recv in case error messages are expected. Intuitively speaking, error predictability demands that any error symbols returned by Recv on input the 'next ciphertext' c can be efficiently predicted given only the ciphertext stream C_S output by Send, the ciphertext stream C_R input to Recv, and the ciphertext c.

As formalized in Definition 4.4 and through the security experiment of Figure 6, we say that a channel provides *error predictability* (ERR-PRE) with respect to an efficient probabilistic *predictor* algorithm Pred if this predictor Pred, given C_R, C_S, and c, accurately outputs the above-mentioned error string with high probability, for every arbitrarily chosen ciphertext c. Put differently, the ERR-PRE experiment declares its adversary to be successful if it ever queries a (counterfeit) ciphertext c that induces Recv to produce different errors from those output by the predictor. Note that the adversary can always learn if it has won by evaluating the winning condition "$\langle m\rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$" itself for the available data.

Definition 4.4 (Error predictability (ERR-PRE))**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel with error space* $\mathcal{E}$*, and let* Pred *be an efficient probabilistic algorithm. We say that* Ch *provides* error predictability (ERR-PRE) with respect to Pred *if for every PPT adversary* $\mathcal{A}$ *playing the experiment* ERR-PRE *defined in Figure 6 against channel* Ch*, the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(1^{\lambda}) = 1\right].$$

The next theorem formalizes the idea that, for the class of error-predictable channels, the generic composition theorem holds (even for channels supporting multiple decryption errors).

```
$\mathsf{Expt}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{A}}(1^\lambda)$:
 1 $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
 2 $\mathsf{win} \leftarrow 0$
 3 $C_S, C_R \leftarrow \varepsilon$
 4 $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
 5 return $\mathsf{win}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m, f)$:
 6 $(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, m, f)$
 7 $C_S \leftarrow C_S \parallel c$
 8 return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
 9 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
10 if $\langle m \rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$ then
11     $\mathsf{win} \leftarrow 1$
12 $C_R \leftarrow C_R \parallel c$
13 return $m$ to $\mathcal{A}$
```

Figure 6: Security experiment for *error predictability* (ERR-PRE) of stream-based channels. We denote by $\langle\cdot\rangle_{\mathcal{E}} \colon (\{0,1\} \cup \mathcal{E})^* \rightarrow \mathcal{E}^*$ the 'projection onto the error space', i.e., the mapping that removes from a string all occurrences that do not belong to the error space $\mathcal{E}$. For instance, let $\mathcal{E} = \{\bot_1, \bot_2\}$ and $m = 01\bot_1 100\bot_2$; then $\langle m \rangle_{\mathcal{E}} = \bot_1\bot_2$.

Theorem 4.5 (INT-CST$\wedge$IND-CPFA$\wedge$ERR-PRE $\Longrightarrow$ IND-CCFA)**.** *Let* Ch = (Init, Send, Recv) *be a (correct) stream-based channel. If* Ch *provides integrity of ciphertext streams, error predictability with respect to a predictor* Pred*, and indistinguishability under chosen plaintext-fragment attacks then it also provides indistinguishability under chosen ciphertext-fragment attacks. Formally, for every efficient* IND-CCFA *adversary $\mathcal{A}$ there exist efficient* INT-CST *adversary $\mathcal{B}$,* ERR-PRE *adversary $\mathcal{C}$, and* IND-CPFA *adversary $\mathcal{D}$ such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}}_{\mathsf{Ch},\mathcal{A}} \leq 2 \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{B}} + 2 \cdot \mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} + \mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch},\mathcal{D}}.$$

Proof. We will consider a sequence of game transitions from the IND-CCFA experiment to the IND-CPFA experiment and bound the difference in probability between each game and its successor in the sequence with the advantage of a specific adversary. For better legibility we will denote the intermediate experiments by $\mathsf{E}^{i,b}_{\mathcal{A}}$ for $i \in \{0, 1, 2\}$ and, with a slight abuse of notation, use the shorthand $\Pr[\mathsf{E}^{i,b}_{\mathcal{A}}]$ to indicate the probability $\Pr[\mathsf{E}^{i,b}_{\mathsf{Ch},\mathcal{A}} = 1]$. In the game transitions we only change the experiment's $\mathcal{O}_{\mathsf{Recv}}$ oracle behavior; the modifications are also shown in Figure 7.

Starting from the IND-CCFA experiment of Figure 3 against $\mathcal{A}$, that we denote $\mathsf{E}^{0,b}_{\mathcal{A}}$, we define a new experiment $\mathsf{E}^{1,b}_{\mathcal{A}}$ which provides the adversary only with the output of the error predictor in case it breaks ciphertext integrity of streams. More precisely, we modify the $\mathcal{O}_{\mathsf{Recv}}$ oracle in $\mathsf{E}^{1,b}_{\mathcal{A}}$ as follows: we add before Lines 13 and 29 of the original experiment a conditional check for $m \notin \mathcal{E}^*$ (before line 13) resp. $m' \notin \mathcal{E}^*$ (before line 29). If either of these conditions evaluates to true, we set a flag bad^b_I and replace m, resp. m', with the output of $\mathsf{Pred}(C_S, C_R, c)$. Let bad^b_I also denote event that the flag bad^b_I is set to true. Note that $\mathsf{E}^{1,b}_{\mathcal{A}}$ and $\mathsf{E}^{0,b}_{\mathcal{A}}$ execute the same instructions as long as bad^b_I does not happen (beyond bookkeeping of C_R in the $\mathsf{sync} = 0$ case which could also be inserted in $\mathsf{E}^{0,b}_{\mathcal{A}}$ without changing its behavior). We can thus assert that $\Pr[\mathsf{E}^{0,b}_{\mathcal{A}} \wedge \neg bad^b_I] = \Pr[\mathsf{E}^{1,b}_{\mathcal{A}} \wedge \neg bad^b_I]$, and hence obtain the bound $\left|\Pr[\mathsf{E}^{0,b}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,b}_{\mathcal{A}}]\right| \leq \Pr[bad^b_I]$ (by the Difference Lemma [Sho06] or the Fundamental Lemma of game playing [BR06]).

We show next how to convert any adversary $\mathcal{A}$ that triggers either event bad^0_I or bad^1_I into an adversary $\mathcal{B}$ that violates the INT-CST security of Ch. Adversary $\mathcal{B}$ initially chooses a bit d uniformly at random and then runs $\mathcal{A}$, answering its queries as follows. If $\mathcal{A}$ queries (m_0, m_1, f) to $\mathcal{O}_{\mathsf{LoR}}$ then $\mathcal{B}$ queries (m_d, f) to its oracle $\mathcal{O}_{\mathsf{Send}}$ and forwards the oracle's answer to $\mathcal{A}$. Similarly $\mathcal{B}$ relays every receiving query c to its oracle $\mathcal{O}_{\mathsf{Recv}}$, obtains a response m, and returns the projection $\langle m \rangle_{\mathcal{E}}$ of m onto the error space $\mathcal{E}$ to $\mathcal{A}$. Note that if the message $\mathcal{A}$ is supposed to obtain were to contain any non-error symbol then it would trigger the bad event. When $\mathcal{A}$ halts, so does $\mathcal{B}$.

As games $\mathsf{E}^{0,b}_{\mathcal{A}}$ and $\mathsf{E}^{1,b}_{\mathcal{A}}$ are the same until bad^b_I we conclude that $\mathcal{B}$ provides a perfect simulation of the games as long as $\mathcal{A}$ does not trigger bad^b_I. Moreover, if $\mathcal{A}$ triggers bad^b_I then $\mathcal{B}$ wins in the INT-CST experiment if it had chosen $d = b$. Thus, we derive the inequality $\mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{B}} \geq \Pr[bad^0_I \wedge d = 0] + \Pr[bad^1_I \wedge d = 1] = \frac{1}{2} \cdot \Pr[bad^0_I] + \frac{1}{2} \cdot \Pr[bad^1_I]$, from which we can bound the advantage of $\mathcal{A}$ in the

```
$\mathcal{O}_{\mathsf{Recv}}(c)$ in $\mathsf{E}^{0,b}_{\mathcal{A}}$:
11 if $\mathsf{sync} = 0$ then
12     $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
13     return $m$ to $\mathcal{A}$
14 else if $C_R \parallel c \preccurlyeq C_S$ then
15     $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
16     $C_R \leftarrow C_R \parallel c$
17     return $\varepsilon$ to $\mathcal{A}$
18 else
19     if $C_R \prec [C_R \parallel c, C_S]$ then
20         $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
21         $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
22         $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
23         $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
24         $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
25     else
26         $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
27     if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
28         $\mathsf{sync} \leftarrow 0$
29     $C_R \leftarrow C_R \parallel c$
30     return $m'$ to $\mathcal{A}$
```

```
$\mathcal{O}_{\mathsf{Recv}}(c)$ in $\mathsf{E}^{1,b}_{\mathcal{A}}$:
1  if $\mathsf{sync} = 0$ then
2      $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
3      $\boxed{\text{if } m \notin \mathcal{E}^* \text{ then}}$
4          $\boxed{bad^b_I \leftarrow \mathsf{true}}$
5          $\boxed{m \leftarrow_{\$} \mathsf{Pred}(C_S, C_R, c)}$
6      $\boxed{C_R \leftarrow C_R \parallel c}$
7      return $m$ to $\mathcal{A}$
8  else if $C_R \parallel c \preccurlyeq C_S$ then
9      $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
10     $C_R \leftarrow C_R \parallel c$
11     return $\varepsilon$ to $\mathcal{A}$
12 else
13     if $C_R \prec [C_R \parallel c, C_S]$ then
14         $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
15         $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
16         $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
17         $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
18         $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
19     else
20         $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
21     if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
22         $\mathsf{sync} \leftarrow 0$
23     $\boxed{\text{if } m' \notin \mathcal{E}^* \text{ then}}$
24         $\boxed{bad^b_I \leftarrow \mathsf{true}}$
25         $\boxed{m' \leftarrow_{\$} \mathsf{Pred}(C_S, C_R, c)}$
26     $C_R \leftarrow C_R \parallel c$
27     return $m'$ to $\mathcal{A}$
```

```
$\mathcal{O}_{\mathsf{Recv}}(c)$ in $\mathsf{E}^{2,b}_{\mathcal{A}}$:
1  if $\mathsf{sync} = 0$ then
2      $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
3      if $m \notin \mathcal{E}^*$ then
4          $bad^b_I \leftarrow \mathsf{true}$
5          $m \leftarrow_{\$} \mathsf{Pred}(C_S, C_R, c)$
6      $\boxed{e \leftarrow_{\$} \mathsf{Pred}(C_S, C_R, c)}$
7      $\boxed{\text{if } m \neq e \text{ then}}$
8          $\boxed{bad^b_E \leftarrow \mathsf{true}}$
9      $C_R \leftarrow C_R \parallel c$
10     $\boxed{\text{return } e \text{ to } \mathcal{A}}$
11 else if $C_R \parallel c \preccurlyeq C_S$ then
12     $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
13     $C_R \leftarrow C_R \parallel c$
14     return $\varepsilon$ to $\mathcal{A}$
15 else
16     if $C_R \prec [C_R \parallel c, C_S]$ then
17         $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
18         $\widetilde{\mathsf{st}_R} \leftarrow \mathsf{st}_R$
19         $(\widetilde{\mathsf{st}_R}, \widetilde{m}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}_R}, \widetilde{c})$
20         $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
21         $m' \leftarrow m \,\%\, [m, \widetilde{m}]$
22     else
23         $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
24     if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
25         $\mathsf{sync} \leftarrow 0$
26     if $m' \notin \mathcal{E}^*$ then
27         $bad^b_I \leftarrow \mathsf{true}$
28         $m' \leftarrow_{\$} \mathsf{Pred}(C_S, C_R, c)$
29     $\boxed{e \leftarrow_{\$} \mathsf{Pred}(C_S, C_R, c)}$
30     $\boxed{\text{if } m' \neq e \text{ then}}$
31         $\boxed{bad^b_E \leftarrow \mathsf{true}}$
32     $C_R \leftarrow C_R \parallel c$
33     $\boxed{\text{return } e \text{ to } \mathcal{A}}$
```

Figure 7: The modifications in the proof of Theorem 4.5 within the $\mathcal{O}_{\mathsf{Recv}}$ oracle in experiments $\mathsf{E}^{0,b}_{\mathcal{A}}$ (equal to the IND-CCA experiment), $\mathsf{E}^{1,b}_{\mathcal{A}}$, and $\mathsf{E}^{2,b}_{\mathcal{A}}$. Lines in $\boxed{\text{frames}}$ in experiment $\mathsf{E}^{i,b}_{\mathcal{A}}$ differ from those in the previous experiment $\mathsf{E}^{i-1,b}_{\mathcal{A}}$. Other lines (on same height) are identical in both experiments. Line numbering in $\mathsf{E}^{0,b}_{\mathcal{A}}$ is as in the IND-CCFA experiment (see Figure 3).

```
$\mathcal{D}^{\mathcal{A},\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot,\cdot)}(1^\lambda)$:
sync $\leftarrow 1$
$C_S, C_R \leftarrow \varepsilon$
$b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}^*_{\mathsf{LoR}}(\cdot,\cdot,\cdot),\mathcal{O}^*_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
return $b'$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{LoR}}(m_0, m_1, f)$:
if $|m_0| \neq |m_1|$ then
    return $\varepsilon$ to $\mathcal{A}$
$c \leftarrow \mathcal{O}_{\mathsf{LoR}}(m_0, m_1, f)$
$C_S \leftarrow C_S \parallel c$
return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(c)$:
if sync $= 0$ then
    $e \leftarrow_\$ \mathsf{Pred}(C_S, C_R, c)$
    $C_R \leftarrow C_R \parallel c$
    return $e$ to $\mathcal{A}$
else if $C_R \parallel c \preccurlyeq C_S$ then
    $C_R \leftarrow C_R \parallel c$
    return $\varepsilon$ to $\mathcal{A}$
else
    $e \leftarrow_\$ \mathsf{Pred}(C_S, C_R, c)$
    if $C_S \not\preccurlyeq C_R \parallel c$ or $e \neq \varepsilon$ then
        sync $\leftarrow 0$
    $C_R \leftarrow C_R \parallel c$
    return $e$ to $\mathcal{A}$
```

Figure 8: IND-CPFA adversary $\mathcal{D}$ simulates the experiment $\mathsf{E}^{2,b}_{\mathcal{A}}$, as in the proof of Theorem 4.5.

IND-CCFA experiment as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}}_{\mathsf{Ch},\mathcal{A}} &= \left|\Pr[\mathsf{E}^{0,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{0,0}_{\mathcal{A}}]\right| \\ &\leq \left|\Pr[\mathsf{E}^{0,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,1}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{1,0}_{\mathcal{A}}] - \Pr[\mathsf{E}^{0,0}_{\mathcal{A}}]\right| \\ &\leq \Pr[bad^1_I] + \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| + \Pr[bad^0_I] \\ &\leq 2 \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{B}} + \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right|.\end{aligned}$$

Observe that in experiment $\mathsf{E}^{1,b}_{\mathcal{A}}$, if bad^b_I is not triggered, the receiving algorithm when fed with a deviating or exceeding ciphertext fragment either outputs error symbols or an empty string. Using the error predictability of the channel wrt. predictor Pred, we can now predict which one of these two cases actually occurs. To this end, we define a variant of $\mathsf{E}^{1,b}_{\mathcal{A}}$, denoted $\mathsf{E}^{2,b}_{\mathcal{A}}$, in which the receiving oracle additionally processes deviating or exceeding ciphertext fragments using the predictor Pred and provides $\mathcal{A}$ with that output. Furthermore, a flag bad^b_E is set if the output of Pred differs from the output of Recv in these cases. See Figure 7 for the precise changes from $\mathsf{E}^{1,b}_{\mathcal{A}}$ to $\mathsf{E}^{2,b}_{\mathcal{A}}$. Let again bad^b_E also denote the event that the flag bad^b_E is set to true. Then $\mathsf{E}^{1,b}_{\mathcal{A}}$ and $\mathsf{E}^{2,b}_{\mathcal{A}}$ execute the same instructions as long as bad^b_E does not happen, and hence their difference in probability is bounded by $\Pr[bad^b_E]$.

Similarly to the previous hop we define an ERR-PRE adversary $\mathcal{C}$ which runs $\mathcal{A}$, chooses a bit d uniformly at random, and simulates games $\mathsf{E}^{i,b}_{\mathcal{A}}$ for $i \in \{1, 2\}$ by relaying $\mathcal{A}$'s queries to its oracles (as above, left-or-right queries (f, m_0, m_1) are first turned into (f, m_d), then sent to $\mathcal{C}$'s oracle). First of all observe that, in the check $m \neq e$ (resp. $m' \neq e$) triggering bad^b_E, it always holds that $m \in \mathcal{E}^*$ (resp. $m' \in \mathcal{E}^*$) and hence $m = \langle m\rangle_{\mathcal{E}}$ resp. $m' = \langle m'\rangle_{\mathcal{E}}$. Thus, if $\mathcal{A}$ triggers bad^b_E, this makes $\mathcal{C}$ win in the ERR-PRE experiment, as $m \neq e$ if and only if $\langle m\rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$ (and likewise for m').

Using a similar argument as above we deduce $\mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} \geq \frac{1}{2} \cdot \Pr[bad^0_E] + \frac{1}{2} \cdot \Pr[bad^1_E]$, which allows us to bound the advantage of $\mathcal{A}$ in the second experiment as follows:

$$\begin{aligned}\left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| &\leq \left|\Pr[\mathsf{E}^{1,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,1}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,0}_{\mathcal{A}}]\right| + \left|\Pr[\mathsf{E}^{2,0}_{\mathcal{A}}] - \Pr[\mathsf{E}^{1,0}_{\mathcal{A}}]\right| \\ &\leq 2 \cdot \mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{C}} + \left|\Pr[\mathsf{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,0}_{\mathcal{A}}]\right|.\end{aligned}$$

We finally observe that the indistinguishability game $\mathsf{E}^{2,b}_{\mathcal{A}}$ can be safely emulated using an IND-CPFA adversary $\mathcal{D}$, as shown in Figure 8. Here $\mathcal{D}$ is granted oracle access to $\mathcal{O}_{\mathsf{LoR}}$ as in the IND-CPFA experiment of Figure 3 and simulates the $\mathsf{IND\text{-}CCFA}$ oracles $\mathcal{O}^*_{\mathsf{LoR}}$ and $\mathcal{O}^*_{\mathsf{Recv}}$ for $\mathcal{A}$. Adversary $\mathcal{D}$ simply relays $\mathcal{O}^*_{\mathsf{LoR}}$ queries to its oracle $\mathcal{O}_{\mathsf{LoR}}$, while it answers queries to $\mathcal{O}^*_{\mathsf{Recv}}$ on its own by invoking the predictor Pred and returning its output.[9] This leads to the following bound:

$$\left|\Pr[\mathsf{E}^{2,1}_{\mathcal{A}}] - \Pr[\mathsf{E}^{2,0}_{\mathcal{A}}]\right| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch},\mathcal{D}}.$$

Combining the bounds implied by the transitions of games above yields the stated security bound. □

Remark 4.6 (Error predictability vs. error simulatability)**.** After the original publication of this work, Barwell et al. introduced the notion of *error simulatability* for subtle authenticated encryption [BPS15b, BPS15a]. Error simulatability is similar in spirit to, but seemingly weaker than, error predictability (the latter requires the existence of an efficient algorithm that outputs *the same* errors produced by Recv while the former only demands that simulated errors be *indistinguishable*[10] from those output by Recv). In fact, since error predictability is defined for a stateful primitive and error simulatability accounts for a stateless primitive, the two notions are incomparable. However, if error simulatability were to be adapted to the streaming setting, it seems plausible that Theorem 4.5 also holds under such weaker requirement. We leave open how to adapt error simulatability to the context of stream-based channels and to investigate further how it relates to error predictability.

5 Stream-Based Channels from AEAD

In this section we demonstrate the feasibility of our security notions by providing a generic construction of stream-based channels which is directly based on the well-established primitive of authenticated encryption with associated data (AEAD) and provides strong security in terms of confidentiality as well as integrity. Although being illustrative rather than definitive, we remark that our construction is quite close to the TLS record protocol. Before specifying the details of our construction we recall the basic properties of AEAD.

5.1 Authenticated Encryption with Associated Data

Originally, authenticated encryption with associated data (AEAD) was introduced as a variant of nonce-based symmetric encryption in [Rog02], however, its syntax can be flexibly adapted to randomized and stateful algorithms. In this paper we stay close to our channel syntax and consider a randomized encryption algorithm. An AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and distinguished error symbol $\perp$ is defined as follows. The probabilistic encryption algorithm Enc takes as input a key $K \in \mathcal{K}$, a message $m \in \{0,1\}^*$, and an associated data string $ad \in \{0,1\}^*$, and outputs a ciphertext $c \in \{0,1\}^*$. The deterministic decryption algorithm Dec takes as input a key $K \in \mathcal{K}$, a ciphertext $c \in \{0,1\}^*$, and an associated data string $ad \in \{0,1\}^*$, and outputs a message $m \in \{0,1\}^*$ or the error symbol $\perp$. We correspondingly write $c \leftarrow_{\$} \mathsf{Enc}_K(ad, m)$ and $m \leftarrow \mathsf{Dec}_K(ad, c)$. Correctness requires that for every key $K \in \mathcal{K}$, every message $m \in \{0,1\}^*$, every associated data string $ad \in \{0,1\}^*$, and every choice of the randomness for the encryption algorithm, it holds that $\mathsf{Dec}_K(ad, \mathsf{Enc}_K(ad, m)) = m$.

[9]We note that $\mathcal{D}$ could actually always downright invoke the error predictor, which would be conceptually closer to the original composition theorem proof for stateful encryption [BKN04]. We however decided to follow the in-sync/out-of-sync case distinction to facilitate comparison with the structure of $\mathcal{O}_{\mathsf{Recv}}$ in $\mathsf{E}^{2,b}_{\mathcal{A}}$.

[10]More precisely, a *subtle AE* scheme is an AE scheme augmented with a 'leakage function' describing how (a specific implementation of) the decryption algorithm reacts when processing invalid ciphertexts. Thus, according to [BPS15b], the simulator's output need not be indistinguishable from the output of the decryption algorithm, but rather from the output of the leakage function.

AEAD security is defined as a combination of confidentiality against passive adversaries ($\mathsf{IND\text{-}CPA}$)[11] and integrity of ciphertexts (AUTH). In the $\mathsf{IND\text{-}CPA}$ experiment the adversary has access to a left-or-right encryption oracle $\mathcal{O}_{\mathsf{LoR}}$. The adversary can query any pair (ad, m_0, m_1) of equal length messages and the oracle returns $c \leftarrow_{\$} \mathsf{Enc}_K(ad, m_b)$ where b is a challenge bit; the adversary's goal is to predict b. We correspondingly define the $\mathsf{IND\text{-}CPA}$ advantage as $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot)} = 1 \mid b = 1] - \Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot)} = 1 \mid b = 0]|$, where the probability is taken over the randomness of the experiment, including $\mathcal{A}$'s randomness.

In the AUTH experiment the adversary has access to an encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ that returns AEAD encryptions of adversarially chosen pairs (ad, m); the adversary's goal is to forge a valid pair (ad^*, c^*) for which no query (ad^*, m^*) has been made such that $\mathsf{Enc}_K(ad^*, m^*)$ has returned c^*. We define the AUTH advantage as $\mathsf{Adv}^{\mathsf{AUTH}}_{\mathsf{AEAD},\mathcal{A}}(\lambda) = \Pr[(ad^*, c^*) \leftarrow_{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}(\cdot)} : \mathsf{Dec}_K(ad^*, c^*) \neq \bot]$, where we implicitly presume that (ad^*, c^*) is fresh.

5.2 Construction Based on Authenticated Encryption with Associated Data

We now propose a generic construction of a stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ from any AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and error symbol $\bot$. For $\mathsf{il}, \mathsf{ol} \in \mathbb{N}$ we assume that AEAD supports encryption of variable-length messages of up to il bits and that the ciphertext output for any such message has bit-length at most $2^{\mathsf{ol}} - 1$. This enables us to encode the length of each AEAD ciphertext with a fixed-size string of ol bits. Our channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ has sending state space $\mathcal{S}_S = \mathcal{K} \times \mathbb{N} \times \{0,1\}^*$, receiving state space $\mathcal{S}_R = \mathcal{K} \times \mathbb{N} \times \{0,1\}^* \times \{0,1\}$, and error space $\mathcal{E} = \{\bot\}$. In Figure 9 we give an algorithmic specification of our construction, and describe it next in detail.

Construction 5.1 (AEAD-based construction $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *Consider an AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ *with key space* $\mathcal{K}$ *and error symbol* $\bot$*. We define* $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *to be the stream-based channel algorithmically specified in Figure 9 and described in detail below.*

Init**.** The initialization algorithm first draws uniformly at random a key K for the AEAD scheme. It then initializes the sending and receiving state respectively as tuples containing key K, a sequence number set to 0, and a message-fragment resp. ciphertext-fragment buffer initially empty; the receiving state also contains a failure flag fail, initially set to 0.

Send**.** The sending algorithm keeps on buffering input message strings until it has collected at least il bits. If sufficiently many bits have been collected, then Send invokes the AEAD encryption algorithm on input message chunks m' of length $|m'| = \mathsf{il}$ and a running sequence number seqno as associated data.[12] The corresponding AEAD ciphertext c' is then prepended with the binary encoding of its size, i.e., the bitstring $len = |c'|$, and the resulting string is appended to the ciphertext string c to be output. In case the Send algorithm was called with the flush flag set to 1, in a final step it also AEAD encrypts any remaining buffered message in the same way, in order to empty the message buffer (this message will potentially contain less than il bits). Note that the size encoding len is a bitstring of fixed length ol and it is not authenticated. Henceforth we may refer to the concatenation of an AEAD ciphertext c' prepended with its size encoding len as a 'block' $B = len \parallel c'$ (see line 9 in Figure 9).

[11] Rogaway [Rog02] actually defines AEAD confidentiality using the stronger $\mathsf{IND\$\text{-}CPA}$ notion, demanding that ciphertexts are indistinguishable from randomly chosen bitstrings, which in particular implies $\mathsf{IND\text{-}CPA}$ security based on a standard left-or-right encryption oracle. We only require $\mathsf{IND\text{-}CPA}$, though, as it is sufficient for our security proof.

[12] A variant construction in the nonce-based setting would use seqno as the encryption nonce and have empty associated data input, as done, e.g., in the TLS 1.3 record protocol from draft-11 [Res15] on. We have chosen the present construction because of its closeness to TLS 1.2 [DR08] and initial drafts of TLS 1.3 (before draft-11), which treat the sequence number as associated data.

```
$\mathsf{Init}(1^\lambda)$:
$K \leftarrow_\$ \mathcal{K}$
$\mathsf{st}_{S,0} = (K, 0, \varepsilon)$
$\mathsf{st}_{R,0} = (K, 0, \varepsilon, 0)$
return $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$
```

```
$\mathsf{Send}(\mathsf{st}_S, m, f)$:
parse $\mathsf{st}_S$ as $(K, \mathsf{seqno}, \mathsf{buf})$
$\mathsf{buf} \leftarrow \mathsf{buf} \parallel m$
$c \leftarrow \varepsilon$
while $|\mathsf{buf}| \geq \mathsf{il}$ do
    $m' \leftarrow \mathsf{buf}[1, \ldots, \mathsf{il}]$
    $\mathsf{buf} \leftarrow \mathsf{buf} \,\%\, m'$
    $c' \leftarrow \mathsf{Enc}_K(\mathsf{seqno}, m')$
    $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
    $B \leftarrow |c'| \parallel c'$ // $|c'| \in \{0,1\}^{\mathsf{ol}}$
    $c \leftarrow c \parallel B$
if $f = 1$ and $\mathsf{buf} \neq \varepsilon$ then
    $c' \leftarrow \mathsf{Enc}_K(\mathsf{seqno}, \mathsf{buf})$
    $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
    $c \leftarrow c \parallel |c'| \parallel c'$ for $|c'| \in \{0,1\}^{\mathsf{ol}}$
    $\mathsf{buf} \leftarrow \varepsilon$
$\mathsf{st}_S \leftarrow (K, \mathsf{seqno}, \mathsf{buf})$
return $(\mathsf{st}_S, c)$
```

```
$\mathsf{Recv}(\mathsf{st}_R, c)$:
parse $\mathsf{st}_R$ as $(K, \mathsf{seqno}, \mathsf{buf}, \mathsf{fail})$
if $\mathsf{fail} = 1$ then
    return $(\mathsf{st}_R, \perp)$
$\mathsf{buf} \leftarrow \mathsf{buf} \parallel c$
$m \leftarrow \varepsilon$
while $|\mathsf{buf}| \geq \mathsf{ol}$ do
    parse $\mathsf{buf}[1, \ldots, \mathsf{ol}]$ as integer $\ell$
    if $|\mathsf{buf}| \geq \mathsf{ol} + \ell$ then
        $len \leftarrow \mathsf{buf}[1, \ldots, \mathsf{ol}]$
        $c' \leftarrow \mathsf{buf}[\mathsf{ol}+1, \ldots, \mathsf{ol}+\ell]$
        $\mathsf{buf} \leftarrow \mathsf{buf} \,\%\, len \parallel c'$
        $m' \leftarrow \mathsf{Dec}_K(\mathsf{seqno}, c')$
        $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
        $m \leftarrow m \parallel m'$
        if $m' = \perp$ then
            $\mathsf{fail} \leftarrow 1$
            break // leave while loop
    else
        break // leave while loop
$\mathsf{st}_R \leftarrow (K, \mathsf{seqno}, \mathsf{buf}, \mathsf{fail})$
return $(\mathsf{st}_R, m)$
```

Figure 9: Generic construction of a stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ from any authenticated encryption with associated data (AEAD) scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and distinguished error symbol $\perp$ which allows to encrypt variable-length messages of up to il bits and for which the ciphertext output has length at most $2^{\mathsf{ol}} - 1$ bits.

Recv. The receiving algorithm outputs an error (without any further state modification) once a first error has emerged from the AEAD decryption algorithm in some previous call (this is indicated by the failure flag set to $\mathsf{fail} = 1$); otherwise, it appends the incoming ciphertext fragment to its buffer. In case enough bits to parse the length field of ol bits were received it does so. Next, it checks whether the buffer contains a complete AEAD ciphertext c' of the indicated length len and, if so, strips it from the buffer, decrypts it (incrementing the sequence number used in the associated data), and appends the result m' to the message m to be output. This process is repeated until there is no full block $B = len \parallel c'$ left in the buffer. However, in case the AEAD decryption algorithm outputs an error, after appending the error symbol $\perp$ to the output message, the Recv algorithm sets the failure flag to 1 and stops parsing further input.

Correctness. For correctness, observe that the Send algorithm generates a ciphertext output which always consists of blocks of ol bits plus the number ℓ of bits binary encoded in these ol bits (at most $2^{\mathsf{ol}} - 1$), where the sequence number seqno is increased with each such block output. Moreover, when called with the flush flag set to 1, the Send algorithm ensures that the entire message stream input so far is written into the ciphertext being output. The Recv algorithm buffers its input and re-establishes the same blocks of $\mathsf{ol} + \ell$ bits as generated by Send, thus also assigning the same sequence number to each block. Since the AEAD decryption algorithm in Recv is called on the same sequence number and exactly the output generated by the encryption algorithm call in Send, we obtain correctness for our generic stream-based channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ by virtue of the correctness of the AEAD scheme.

5.3 Security Analysis

Our generic stream-based channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ from Construction 5.1 provides indistinguishability under chosen plaintext-fragment attacks (IND-CPFA), integrity of ciphertext streams (INT-CST), and error predictability (ERR-PRE), given that the underlying authenticated encryption with associated data scheme AEAD provides indistinguishability under chosen plaintext attacks (IND-CPA) and authenticity (AUTH) as defined in Section 5.1. Using Theorem 4.5 we can moreover infer that it also provides indistinguishability under chosen ciphertext-fragment attacks (IND-CCFA).

Theorem 5.2 (IND-CPFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 5.1 provides indistinguishability under chosen plaintext-fragment attacks (*IND-CPFA*) if the authenticated encryption with associated data scheme* AEAD *provides indistinguishability under chosen plaintext attacks (*IND-CPA*). Formally, for every efficient* IND-CPFA *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD}}$ *there exists an efficient* IND-CPA *adversary* $\mathcal{B}$ *against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{B}}(\lambda).$$

Proof. We reduce the IND-CPFA security of $\mathsf{Ch}_{\mathsf{AEAD}}$ to the IND-CPA security of AEAD by constructing from an efficient adversary $\mathcal{A}$ against the former property an efficient adversary $\mathcal{B}$ against the latter property. In order to simulate the $\mathcal{O}_{\mathsf{LoR}}$ oracle for $\mathcal{A}$ we let $\mathcal{B}$ perform the buffering and sending procedure as defined for Send in Figure 9, keeping two buffers for the two message inputs m_0 and m_1 from $\mathcal{A}$ and a sequence number. As the buffering behavior and sending procedure only depends on the length of the input message to Send but not its content, the message blocks to be encrypted using the AEAD scheme are treated identically for either the m_0 or the m_1 buffer. This allows $\mathcal{B}$ to replace the encryption operations by calls to its encryption oracle in the IND-CPA game with the according blocks (of same size) both from the m_0 and m_1 buffer and the (always coinciding) sequence number. This results in a single ciphertext which $\mathcal{B}$ can then process further, as defined in the Send algorithm, to provide the output ciphertext to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs its guessed bit b', we let $\mathcal{B}$ output the same bit as its guess.

Assume $\mathcal{A}$ is successful. Since $\mathcal{B}$ perfectly simulates the $\mathcal{O}_{\mathsf{LoR}}$ oracle for $\mathcal{A}$, it also wins in the IND-CPFA game. □

Theorem 5.3 (INT-CST security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 5.1 provides integrity of ciphertext streams (*INT-CST*) if the authenticated encryption with associated data scheme* AEAD *provides authenticity (*AUTH*). Formally, for every efficient* INT-CST *adversary* $\mathcal{A}$ *against* $\mathsf{Ch}_{\mathsf{AEAD}}$ *there exists an efficient* AUTH *adversary* $\mathcal{B}$ *against* AEAD *such that*

$$\mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{AUTH}}_{\mathsf{AEAD},\mathcal{B}}(\lambda).$$

Proof. Recall that the receiving algorithm Recv of our channel construction processes the ciphertext stream by identifying blocks $B_1, B_2, \ldots$ with $B_i = len_i \parallel c'_i$ where c'_i is an AEAD ciphertext and len_i is the (fixed-length) binary encoding of its size $|c'_i|$. The message fragments output by Recv are obtained by concatenating the AEAD decryptions m'_i of the so identified ciphertexts c'_i. In particular, Recv produces some non-trivial output if and only if it processes at least a *full block* B_i. The main observation is that, in order to break the INT-CST property of $\mathsf{Ch}_{\mathsf{AEAD}}$, an adversary must submit to $\mathcal{O}_{\mathsf{Recv}}$ a non-genuine (i.e., deviating or exceeding) ciphertext stream whose non-genuine part contains a full valid block $B^* = len^* \parallel c'^*$. More precisely, the AEAD decryption of c'^* with the current sequence number seqno as associated data must yield some valid message m^*. Now, since the scheme increases seqno before each AEAD encryption, no associated data is ever repeated. Moreover, by hypothesis the block B^* deviates from the genuine ciphertext stream. Thus (seqno, c'^*) is an AEAD forgery.

We now formalize this intuition. Let $\mathcal{A}$ be an adversary attacking the $\mathsf{INT\text{-}CST}$ of channel $\mathsf{Ch}_{\mathsf{AEAD}}$. We build an adversary $\mathcal{B}$ which runs $\mathcal{A}$ internally as a black box and breaks the AUTH property of AEAD as long as $\mathcal{A}$ is successful against the $\mathsf{INT\text{-}CST}$ property of $\mathsf{Ch}_{\mathsf{AEAD}}$. Adversary $\mathcal{B}$ emulates the channel construction (see Figure 9) by forwarding AEAD encryptions to the oracle $\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot)$ provided in the AUTH security experiment and by performing the buffering steps on its own. For this it keeps buffer strings buf_S, buf_R and a sequence number seqno, initialized to the empty strings and to zero respectively. It also keeps lists $\mathbf{m}'$ and $\mathbf{c}'$ for bookkeeping of sent AEAD messages and ciphertexts respectively, as well as a string C_R in which it registers the received (stream) ciphertext fragments.

$\mathcal{B}$ answers $\mathcal{A}$'s queries as follows.

- When $\mathcal{A}$ poses a *sending query* (m, f), $\mathcal{B}$ appends m to the buffer buf_S, initializes an empty ciphertext c, and repeats the steps of instructions 4–10 from Figure 9. In particular, $\mathcal{B}$ performs the encryption steps by querying $\mathcal{O}_{\mathsf{Enc}}$ on tuples (ad, m') where $ad = \mathsf{seqno}$ is a running sequence number, and registers the AEAD messages m' and ciphertexts c' in the lists $\mathbf{m}'$ and $\mathbf{c}'$, respectively, according to the order in which they are processed. If the flush flag is set to $f = 1$, $\mathcal{B}$ executes one more encryption (oracle call) using as m' the remaining buffer. Finally it returns c to $\mathcal{A}$.

- When $\mathcal{A}$ poses a *receiving query* c, the reduction updates buffer buf_R and string C_R by appending c to both of them and checks if c is genuine by comparing the components of $\mathbf{c}'$ with the blocks $B_i = len_i \parallel c'_i$ contained in C_R.

  If c is genuine, $\mathcal{B}$ traverses the buffer buf_R and identifies the blocks B_i corresponding to the sent AEAD ciphertexts. Recall that for each block $B_i = len_i \parallel c'_i$ the AEAD ciphertext c'_i is registered in the list $\mathbf{c}'$ and, correspondingly, its decryption m' is registered in the list $\mathbf{m}'$. Thus, $\mathcal{B}$ can for each identified ciphertext c'_i recover the decryption m'_i. After this, $\mathcal{B}$ removes the identified blocks $B_i = len_i \parallel c'_i$ from the buffer buf_R and concatenates all corresponding messages m'_i in the same order they appear in $\mathbf{m}'$, obtaining a string m; finally $\mathcal{B}$ gives m to $\mathcal{A}$.

  If c is not genuine, $\mathcal{B}$ first performs the procedure above, but using instead of c its longest genuine prefix $\widetilde{C}$ *that contains only full blocks* B_i. Note that after this step the buffer buf_R will be empty (because it only contained full blocks and all these are processed). Afterwards $\mathcal{B}$ tries to extract a forgery from the remaining part of c and potentially subsequent queries: if $c \,\%\, \widetilde{C}$ contains a full block B^* then $\mathcal{B}$ immediately extracts a forgery, otherwise it keeps answering with the empty string all subsequent receiving queries until it gets enough ciphertext bits to extract a forgery. As soon as the buffer C_R is augmented with at least a non-genuine full block $B^* = len^* \parallel c'^*$, the reduction outputs as forgery (ad^*, c'^*) with $ad^* = \mathsf{seqno} + 1$ and halts. Note that $\mathsf{Ch}_{\mathsf{AEAD}}$ by construction from the first occurring AEAD error on only outputs outputs errors. Hence, if $\mathcal{A}$ succeeds at all in breaking integrity, then it will be with the first deviating ciphertext, which consequently $\mathcal{B}$ outputs as its forgery attempt.

It is immediate to see that $\mathcal{B}$ performs a sound simulation of the $\mathsf{INT\text{-}CST}$ experiment. Indeed, for answering sending queries it executes the same instructions as Send (only the AEAD encryption is replaced with an oracle call to $\mathcal{O}_{\mathsf{Enc}}$, however, the AEAD encryption takes place within the oracle). Although $\mathcal{B}$ lacks a decryption oracle and, thus, cannot process $\mathcal{A}$'s receiving queries, it can answer all genuine queries since, for these, the AEAD decryption is correct. In the same way $\mathcal{B}$ can recover the longest genuine message fragment underlying the first non-genuine query. Regarding non-genuine decryption queries, $\mathcal{B}$ can either extract an AEAD forgery, or trivially answer by returning an empty message and waiting for more ciphertext bits.

It remains to show that if $\mathcal{A}$ breaks the $\mathsf{INT\text{-}CST}$ property of $\mathsf{Ch}_{\mathsf{AEAD}}$ then $\mathcal{B}$ is successful in the AUTH game against AEAD. Let c denote $\mathcal{A}$'s first out-of-sync query, let $\widetilde{c}$ be the longest in-sync prefix of c, and let m and $\widetilde{m}$ be the message fragments that Recv would output on input c and $\widetilde{c}$ respectively in the real execution of the $\mathsf{INT\text{-}CST}$ experiment.

Assume first that $\mathcal{A}$ is successful with its first non-genuine query to $\mathcal{O}_{\mathsf{Recv}}$: we thus have $m\%[m, \widetilde{m}] \notin \mathcal{E}^*$. Suppose that the (genuine) ciphertext stream that Recv would process up to $\widetilde{c}$ contains the first i AEAD blocks $B_1, \ldots, B_i$ sent. By construction $\widetilde{m} = m'_1 \parallel \cdots \parallel m'_i$ where each m'_i is the AEAD decryption of c'_i. Then the ciphertext fragment $c \,\%\, (B_1 \parallel \cdots \parallel B_i)$ contains as a prefix a full block $B^* = len^* \parallel c'^*$ such that c'^* with associated data $\mathsf{seqno}^* = i + 1$ AEAD-decrypts to $m^* \neq \bot$, otherwise we would have either $m \,\%\, [m, \widetilde{m}] \in \mathcal{E}$ or $m \,\%\, [m, \widetilde{m}] = \varepsilon$, against our hypothesis.

The argument above easily extends to the general case in which $\mathcal{A}$ poses several non-genuine queries to $\mathcal{O}_{\mathsf{Recv}}$ before breaking the $\mathsf{INT\text{-}CST}$ security of $\mathsf{Ch}_{\mathsf{AEAD}}$ by letting $\mathcal{O}_{\mathsf{Recv}}$ create some non-trivial output: $\mathcal{B}$ simply keeps buffering and answering queries with an empty string until it collects enough ciphertext bits to form a full block $B^* = len^* \parallel c'^*$ (here 'enough' means $\mathsf{ol} + len^*$ bits).

In both cases, once $\mathcal{B}$ obtains sufficiently many ciphertext bits to isolate a block B^*, it stops the simulation and returns as valid AEAD forgery the pair (seqno^*, c'^*). □

Theorem 5.4 (Error predictability of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}}$ from Construction 5.1 provides error predictability ($\mathsf{ERR\text{-}PRE}$), with respect to the predictor Pred given in the proof of the theorem, if the authenticated encryption with associated data scheme AEAD provides authenticity (AUTH). Formally, for every efficient $\mathsf{ERR\text{-}PRE}$ adversary $\mathcal{A}$ against $\mathsf{Ch}_{\mathsf{AEAD}}$ and predictor Pred there exists an efficient AUTH adversary $\mathcal{B}$ against AEAD such that*

$$\mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch}_{\mathsf{AEAD}},\mathsf{Pred},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{AUTH}}_{\mathsf{AEAD},\mathcal{B}}(\lambda).$$

Proof. We start by defining the predictor algorithm Pred. On input $C_S \in \{0,1\}^*$, $C_R \in \{0,1\}^*$, and $c \in \{0,1\}^*$ the predictor first computes $C'_R \in \{0,1\}^*$ by removing from $C_R \parallel c$ the longest prefix with C_S which only consists of complete blocks containing each a length field (of ol bits, where ol is determined by the AEAD scheme) followed by as many bits as binary encoded in that length field. In case C'_R contains a complete block (length field plus encoded number of bits), Pred outputs the distinguished error symbol $\bot$ of the AEAD scheme, otherwise it outputs the empty string ε.

Now we reduce the error predictability $\mathsf{ERR\text{-}PRE}$ of $\mathsf{Ch}_{\mathsf{AEAD}}$ to the AUTH security of AEAD by turning any efficient adversary $\mathcal{A}$ that distinguishes the output of Recv from the output of Pred into an efficient adversary $\mathcal{B}$ against the AUTH property of the AEAD scheme. Initially, $\mathcal{B}$ sets $C_S \leftarrow \varepsilon$, $C_R \leftarrow \varepsilon$. As in the previous two proofs, it simulates the oracle $\mathcal{O}_{\mathsf{Send}}$ for $\mathcal{A}$ using the encryption oracle in the AUTH game and furthermore appends the obtained result c to C_S.

For simulating the $\mathcal{O}_{\mathsf{Recv}}$ oracle for $\mathcal{A}$, adversary $\mathcal{B}$ appends c to C_R and returns the empty string ε to $\mathcal{A}$ as long as the ciphertext fragments provided are in sync ($C_R \parallel c \preccurlyeq C_S$). When $\mathcal{A}$ provides ciphertext fragments such that the receiving buffer at some point contains a full block (consisting of an encoded length and the ciphertext of that length) which at some point deviates from the genuine ciphertext stream (i.e., was never output by $\mathcal{B}$'s simulation of $\mathcal{O}_{\mathsf{Send}}$), $\mathcal{B}$ outputs the ciphertext of this block along with the current sequence number value as associated data field as its forgery in the AUTH game and stops.

Note that $\mathcal{B}$ perfectly simulates the $\mathsf{ERR\text{-}PRE}$ experiment for $\mathcal{A}$, as by correctness of $\mathsf{Ch}_{\mathsf{AEAD}}$ and the buffering behavior of Recv, the first point where Recv could output an error symbol is when it received a complete ciphertext block which deviates from the ciphertext stream generated by Send. Up to this point, also Pred will not have output an error, so that $\mathcal{A}$ cannot have won yet.

Assume $\mathcal{A}$ wins at the point where the input ciphertext c completes the first deviating ciphertext block input to Recv. As this requires that $\langle m \rangle_{\mathcal{E}} \neq \mathsf{Pred}(C_S, C_R, c)$ but the output of Pred will be $\bot$, this means that the ciphertext block (and the according sequence number as associated data) decrypts under the AEAD scheme to a valid message (and not the distinct error symbol). Hence, this output constitutes a valid forgery and $\mathcal{B}$ thus also wins in the AUTH game. Furthermore, note that $\mathcal{A}$ cannot win in the

$\mathsf{ERR\text{-}PRE}$ experiment with a later call to its $\mathcal{O}_{\mathsf{Recv}}$ oracle, because after the first error occurred, both Recv and Pred consistently output $\perp$ for any further ciphertext fragment input c. $\square$

Applying Theorem 4.5 we can now deduce the following corollary.

Corollary 5.5 ($\mathsf{IND\text{-}CCFA}$ security of $\mathsf{Ch}_{\mathsf{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch}_{\mathsf{AEAD}}$ *from Construction 5.1 provides indistinguishability under chosen ciphertext-fragment attacks (*$\mathsf{IND\text{-}CCFA}$*) if the authenticated encryption with associated data scheme* AEAD *provides indistinguishability under chosen plaintext attacks (*$\mathsf{IND\text{-}CPA}$*) and authenticity (*AUTH*).*

Remark 5.6. Our $\mathsf{Ch}_{\mathsf{AEAD}}$ construction can be easily modified to produce a channel with multiple error messages (occurring with non-negligible probability) which still permits the definition of a suitable error predictor and which thus remains amenable to the application of our composition theorem. Consider, e.g., a variant of $\mathsf{Ch}_{\mathsf{AEAD}}$ which extends the length field by one bit, but which outputs an error symbol $\perp_2$ different from the AEAD scheme's error symbol $\perp$ when the leading bit in this length field is set to 1. Although an adversary can easily make the Recv algorithm output either of the new channel's two error symbols at will, observe that it is also still easy to define a suitable error predictor for such a channel.

Length fields and potential error conditions arising from malicious encodings are commonly encountered in real-world channel schemes (e.g., TLS with its `bad_record_mac` and `record_overflow` error messages). The above artificial variant of $\mathsf{Ch}_{\mathsf{AEAD}}$ provides an illustrative example of how our approach of using an error predictor in the composition theorem allows us to resurrect the classical composition result in the stream-based setting, even for channels having multiple errors which can all occur with non-negligible probabilities (recall that the corresponding composition theorem in the much simpler atomic setting in [BDPS14] required all but one of the errors to occur with negligible probability).

Remark 5.7. The security of our stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}}$ is based on the $\mathsf{IND\text{-}CPA}$ and AUTH properties of the underlying AEAD scheme, and the AUTH notion [Rog02] is given in the single-error setting. Thus, while our results establish the security of $\mathsf{Ch}_{\mathsf{AEAD}}$ in the multiple-error setting, we still assume that the AEAD scheme in use produces a unique error (or, in the terminology of [BPS15b], that the AEAD scheme is implemented *ideally*). We conjecture that our results also hold in a subtle-AE–like model for the underlying AEAD scheme, and in particular assuming error simulatability (see Remark 4.6).

5.4 A Note on the TLS Record Protocol

As discussed earlier, the Transport Layer Security (TLS) record protocol implements a stream-based channel whose complete analysis as such lies outside of the scope of this work. However we do pause to note that our Construction 5.1 of a stream-based channel based on authenticated encryption with associated data is conceptually close to the TLS record protocol when using an AEAD scheme as specified for TLS version 1.2 [DR08, Section 6.2.3.3] and in the current (as of December 2017) draft for TLS version 1.3 [Res17, Section 5]: the record protocol also incorporates a sequence number which is authenticated but not sent on the wire and a length field which is sent and authenticated in TLS 1.2 (and which is sent but not authenticated in the draft TLS 1.3).[13] However, the TLS record protocol in version 1.2 additionally includes a 2-byte version number and a 1-byte content type; these are both sent and authenticated in the associated data. Moreover, the AEAD schemes used are considered to be nonce-based, with the TLS 1.3 draft specifying how the nonce is formed and TLS 1.2 leaving the exact nonce generation to be specified

[13]That is, our approach of using a length field which is sent on the wire, but not part of the authenticated associated data of the AEAD ciphertext conforms with the approach adopted in the TLS 1.3 draft. In contrast to our approach, the TLS 1.3 draft implicitly authenticates the sequence number by letting it enter the AEAD nonce rather than explicitly authenticating it in the associated data field.

by the particular cipher suite in use. TLS (in both versions) furthermore specifies padding mechanisms and TLS 1.3 uses a double-header structure for backwards compatibility reasons.

The content type field in particular allows TLS to multiplex data streams for different purposes within a single connection stream, as TLS 1.2 does for the Handshake Protocol, the Alert Protocol, the ChangeCipherSpec protocol, and the Application protocol. While our model does not capture multiplexing several message streams into one ciphertext stream, it can be augmented to do so. This brings additional complexity and is an avenue for future work.

6 Atomic-Message Channels Supporting Fragmentation

Many application layer protocols rely on a stream-based transport protocol like TCP, as the data they transmit is inherently stream-based (like in audio or video streaming applications) or may consist of individual messages that are too large to be written to, or read from, the transport protocol in one go (as in HTTP transfers of large files). For such applications, our security model appropriately captures security guarantees to be expected from a cryptographic stream-based transport protocol or channel such as TLS.

Other application layer protocols, however, are inherently message-based (e.g., chat protocols or header transmission in HTTP) and might crucially demand that only messages that are guaranteed to be authentic and complete (i.e., non-truncated) are delivered. Most secure transport protocols in use (e.g., TLS), though, are stream-based in nature and might deliver input messages—which they consider as *fragments*—in several parts, both in real-world implementations and in our model. Unwary processing of message fragments on the receiver's side might thus break, and has broken in the past, the security of message-dependent application layer protocols (see, e.g., [SP13, BDF$^+$14]). This raises the following safety question: How can a reliable and secure transport channel for *atomic messages* be provided on top of a secure *stream-based* channel in order to protect message-dependent application protocols from misinterpreting *partial* messages on the receiver's side as *complete* ones?

Note that previous works on channels do not provide a satisfying solution to this problem. For instance, while Bellare et al. [BKN04] and the follow-up works (e.g., [KPB03, PRS11, JKSS12, BDPS14, BSWW13]) consider the transport of atomic messages, their model lacks potential fragmentation on the network. Boldyreva et al. [BDPS12] made a first step further in this direction by considering confidentiality of channels that treat messages atomically at the sender's side and allow for fragmentation on the network. Integrity in this setting was later, and concurrently to this work, defined by Albrecht et al. [ADHP16]. Still, while their notions approach capturing atomic-message channels over fragmented transport, they do not provide an answer to the question of how to achieve such from a given stream-based channel.

To fill this gap we first introduce the notion of *atomic-message channels supporting fragmentation* which covers schemes that transport atomic messages in a secure way over a potentially fragmenting network. While our (strong) confidentiality and integrity notions are similar in spirit to those defined by Boldyreva et al. [BDPS12] and Albrecht et al. [ADHP16], our syntax already intrinsically encodes an atomic-message interface both for the input on the sender's side as well as for the output on the receiver's side. Beyond that, we further define the corresponding weaker variants of chosen-plaintext confidentiality and plaintext integrity for atomic-message channels supporting fragmentation.

In a second step we propose a generic way to safely transport atomic messages using a stream-based channel. Our strategy is to add an encoding layer that turns a message sequence into a stream of bits and allows to recover from that bit stream the original message sequence; then, we simply transmit the obtained bit stream through any secure stream-based channel.

6.1 Syntax and Functionality

The syntax of atomic-message channels supporting fragmentation for the sending algorithm aSend reverts back to the classical setting with an atomic message input and an atomic ciphertext output, obviating the need for a flush flag. Note also that we therefore use the vector notation where appropriate, e.g., $\mathbf{m}[1,\ldots,i]$ then refers to the first i entries of the vector $\mathbf{m}$ (and not the first i bits of the string m).

To capture ciphertext fragmentation on the underlying network we allow the receiving algorithm aRecv to take a ciphertext fragment as input, but we syntactically require it to output clearly separated atomic messages (instead of chunks of a message stream where the boundaries of individual chunks have no inherent meaning). As a ciphertext fragment might contain more than one original ciphertext output by aSend, we need to allow aRecv to output not only a single message but a vector of messages (which may be empty). In contrast to stream-based channels here we consider a generic message space $\mathcal{M} \subseteq \{0,1\}^*$ instead of fixing $\mathcal{M} = \{0,1\}^*$.

Definition 6.1 (Syntax of atomic-message channels supporting fragmentation)**.** *An* atomic-message channel supporting fragmentation $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *with associated message space $\mathcal{M}$, sending and receiving state space $\mathcal{S}_S$ resp. $\mathcal{S}_R$, and error space $\mathcal{E}$ consists of three efficient algorithms:*

- aInit*. On input a security parameter 1^λ, this probabilistic algorithm outputs initial states $\mathsf{st}_{S,0} \in \mathcal{S}_S$, $\mathsf{st}_{R,0} \in \mathcal{S}_R$ for the sender and the receiver, respectively. We write $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow_\$ \mathsf{aInit}(1^\lambda)$.*

- aSend*. On input a state $\mathsf{st}_S \in \mathcal{S}_S$ and a message $m \in \mathcal{M}$, this (possibly) probabilistic algorithm outputs an updated state $\mathsf{st}'_S \in \mathcal{S}_S$ and a ciphertext $c \in \{0,1\}^*$. We write $(\mathsf{st}'_S, c) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_S, m)$.*

- aRecv*. On input a state $\mathsf{st}_R \in \mathcal{S}_R$ and a ciphertext fragment $c \in \{0,1\}^*$, this deterministic algorithm outputs an updated state $\mathsf{st}'_R \in \mathcal{S}_R$ and a (potentially empty) vector of messages $(m_1,\ldots,m_\ell) \in (\mathcal{M} \cup \mathcal{E})^*$. We write $(\mathsf{st}'_R, (m_1,\ldots,m_\ell)) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$.*

In the rest of the paper we use the shorthand *atomic-message channels* to indicate atomic-message channels supporting fragmentation. We also use the same vector notation as for stream-based channels from Section 3 and correspondingly write, e.g., $(\mathsf{st}_S, \mathbf{c}) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_{S,0}, \mathbf{m})$ to indicate that the sending algorithm is invoked sequentially on input the components of $\mathbf{m}$ and that it outputs as ciphertexts the components of $\mathbf{c}$.

Intuitively, correctness requires that an atomic-message channel recovers the sequence of messages sent as long as the entire sequence of ciphertexts sent is received completely, independently of its fragmentation. It also requires that, if any *prefix* of the sequence of sent ciphertexts is processed at the receiver, then the corresponding prefix of the sent message sequence is recovered completely.

Definition 6.2 (Correctness of atomic-message channels)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel. We say that* aCh *provides* correctness *if for all choices of the randomness for algorithms* $\mathsf{aInit}, \mathsf{aSend}$ *and* aRecv*, all $\ell, \ell' \geq 0$, all message vectors $\mathbf{m} \in \mathcal{M}^\ell$, all sending output sequences $(\mathsf{st}_{S,\ell}, \mathbf{c}) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_{S,0}, \mathbf{m})$, all ciphertext-fragment vectors $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$, all receiving output sequences $(\mathsf{st}'_{R,\ell'}, \mathbf{m}') \leftarrow \mathsf{aRecv}(\mathsf{st}_{R,0}, \mathbf{c}')$, and every $0 \leq i \leq \ell$, we have*

$$\| \mathbf{c}[1,\ldots,i] \preccurlyeq \| \mathbf{c}' \preccurlyeq \| \mathbf{c} \Longrightarrow \mathbf{m}[1,\ldots,i] \preccurlyeq \mathbf{m}' \preccurlyeq \mathbf{m}.$$

Note that, in contrast to [BDPS12, ADHP16], for correctness in the above setting we do not demand that already $\| \mathbf{c}[1,\ldots,i] \preccurlyeq \| \mathbf{c}'$ implies $\mathbf{m}[1,\ldots,i] \preccurlyeq \mathbf{m}'$, even if $\| \mathbf{c}' \not\preccurlyeq \| \mathbf{c}$. As we already discussed in the streaming setting (cf. Remark 3.4), this would encode a certain amount of robustness in an adversarial setting which is concerned with security rather than correctness.

```
$\mathsf{Expt}^{\mathsf{aIND\text{-}atk},b}_{\mathsf{aCh},\mathcal{A}}(1^\lambda)$:
$(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{aInit}(1^\lambda)$
$\mathsf{sync} \leftarrow 1$
$i \leftarrow 0$, $j \leftarrow 1$
$\mathbf{M_S} \leftarrow ()$, $\mathbf{C_S} \leftarrow ()$
$\mathbf{M_R} \leftarrow ()$, $C_R \leftarrow \varepsilon$
$b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{LoR}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
return $b'$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{LoR}}(m_0, m_1)$:
if $|m_0| \neq |m_1|$ then
    return $\varepsilon$ to $\mathcal{A}$
$(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_S, m_b)$
$i \leftarrow i + 1$
$\mathbf{M_S}[i] \leftarrow m_b$, $\mathbf{C_S}[i] \leftarrow c$
return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
$(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
$C_R \leftarrow C_R \parallel c$
$\mathbf{M_R} \leftarrow \mathbf{M_R} \parallel \mathbf{m}$
if $\mathsf{sync} = 0$ then // already out-of-sync
    return $\mathbf{m}$ to $\mathcal{A}$
else if $C_R \preccurlyeq \parallel \mathbf{C_S}$ then // still in-sync
    return $()$ to $\mathcal{A}$
else
    while $j \leq i$ and $\parallel \mathbf{C_S}[1, \ldots, j] \preccurlyeq C_R$ and $\mathbf{M_S}[1, \ldots, j] \preccurlyeq \mathbf{M_R}$
        do $j \leftarrow j + 1$
    if $j \leq i$ or $|\mathbf{M_R}| > i$ then
        // deviation, or exceeding portion produces output
        $\mathsf{sync} \leftarrow 0$
    if $j > |\mathbf{M_R}|$ then
        $\mathbf{m}' \leftarrow ()$
    else
        $\mathbf{m}' \leftarrow \mathbf{M_R}[j, \ldots, |\mathbf{M_R}|]$
    return $\mathbf{m}'$ to $\mathcal{A}$
```

Figure 10: Security experiment for *confidentiality* ($\mathsf{aIND\text{-}atk}$ with $\mathsf{atk} \in \{\mathsf{CPA}, \mathsf{CCFA}\}$) of atomic-message channels. A CPA-attacker only has access to the oracle $\mathcal{O}_{\mathsf{LoR}}$.

6.2 Security

In this section we formalize confidentiality and integrity notions for atomic-message channels.

Confidentiality. In order to translate the standard confidentiality requirements against chosen-plaintext attacks and chosen-ciphertext attacks to the setting of atomic-message channels supporting fragmentation we formulate the notions of atomic-message indistinguishability under chosen-plaintext attacks (aIND-CPA) as well as under chosen ciphertext-fragment attacks (aIND-CCFA). The former provides the adversary with a left-or-right sending oracle defined in the natural way. The latter essentially transcribes the IND-sfCFA notion by Boldyreva et al. [BDPS12] to our setting for atomic-message channels. Briefly, the decryption mechanism of our aIND-CCFA notion returns to the adversary all non-genuine message blocks output on receiving the first deviating ciphertext fragment and all follow-up calls.

In more detail, the adversary is provided with a left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$ and, in the aIND-CCFA experiment, with a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$. Left-or-right queries do not include a flush flag; this is a consequence of the syntax. A major difference with the streaming setting is that, since the receiving algorithm outputs atomic-message sequences rather than portions of a stream, synchronization is lost at the ciphertext boundaries (similarly to the case of symmetric encryption supporting fragmentation [BDPS12]). That is, the exact point where synchronization is lost is not necessarily aligned with its counterpart in the streaming setting. However, in contrast to symmetric encryption supporting fragmentation, here suppressing from the received message sequence the (vector) prefix covered by correctness would be inaccurate, as we explain next.

Suppose that $\mathcal{A}$ causes $\mathcal{O}_{\mathsf{LoR}}$ to send messages $m_1^b, \ldots, m_i^b$ and obtains challenge ciphertexts $c_1, \ldots, c_i$. For the sake of exposition let us make the first out-of-sync ciphertext coincide with the first receiving query and suppose that $\mathcal{A}$ submits to $\mathcal{O}_{\mathsf{Recv}}$ a single ciphertext fragment c^* that contains $c_1 \parallel \cdots \parallel c_{j-1}$ as a prefix, for some $j \leq i$, but deviates from c_j onwards. Correctness then does not impose any requirement on the decryption $\mathbf{m}^*$ of the *adversarially* chosen c^*: it may contain none of the messages sent, but it may also contain the full sequence $(m_1^b, \ldots, m_{j-1}^b)$. Thus, despite c^* being out-of-sync, giving $\mathcal{A}$ the full decryption of c^* could lead to trivial wins. However, suppressing from $\mathbf{m}^*$ its first $j-1$ components may hide non-genuine messages, excluding valid attacks from being caught. Intuitively, we want to suppress from the sequence of received messages only the longest genuine prefix. This intuition explains the working principle of the oracle $\mathcal{O}_{\mathsf{Recv}}$, which answers the first out-of-sync query by returning the sequence obtained from $\mathbf{M_R}$ by stripping off the longest genuine prefix $\mathbf{M_S}[1, \ldots, j-1]$ (see lines 21ff. in Figure 10), and subsequent out-of-sync queries by returning the full output of aRecv. As for the streaming setting, we define synchronization to be lost only on an exceeding ciphertext fragment if that fragment produces (non-empty) output.

Definition 6.3 ($\mathsf{aIND\text{-}CPA}$ and $\mathsf{aIND\text{-}CCFA}$ Security)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel supporting fragmentation and* $\mathsf{Expt}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aIND\text{-}atk},b}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* b *be defined as in Figure 10, where* atk *is a placeholder for either* CPA *or* CCFA*. Within the experiment the adversary* $\mathcal{A}$ *is given access to a left-or-right sending oracle* $\mathcal{O}_{\mathsf{LoR}}$ *and, in the case of* $\mathsf{aIND\text{-}CCFA}$ *security, a receiving oracle* $\mathcal{O}_{\mathsf{Recv}}$*. We say that* aCh *provides* atomic-message indistinguishability under chosen-plaintext attacks*, respectively,* chosen ciphertext-fragment attacks *(*$\mathsf{aIND\text{-}CPA}$ *resp.* $\mathsf{aIND\text{-}CCFA}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathsf{Adv}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aIND\text{-}atk}}(\lambda) := \Pr\left[\mathsf{Expt}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aIND\text{-}atk},1}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aIND\text{-}atk},0}(1^\lambda) = 1\right].$$

Integrity. As for confidentiality we adapt the integrity notions from the streaming setting and define (atomic-message) integrity of plaintexts ($\mathsf{aINT\text{-}PTXT}$) and integrity of ciphertext streams ($\mathsf{aINT\text{-}CST}$). The idea behind plaintext integrity is that the adversary wins if it manages to make the receiver output a valid message sequence that differs from the sequence of messages that have been sent. In integrity of ciphertext streams, instead, the adversary wins if it submits a modified ciphertext sequence for decryption whose deviating part produces some valid messages. Again, our $\mathsf{aINT\text{-}CST}$ notion is analogous to the corresponding $\mathsf{INT\text{-}sfCTF}$ notion by Albrecht et al. [ADHP16] for symmetric encryption supporting fragmentation, proposed concurrently to this work here.

In both integrity experiments the adversary is provided with a sending oracle $\mathcal{O}_{\mathsf{Send}}$ and a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ that it can query on arbitrary messages, respectively, ciphertext fragments. The $\mathsf{aINT\text{-}PTXT}$ experiments declares $\mathcal{A}$ successful if the sequence $\mathbf{M_R}$ of received messages deviates from the sequence $\mathbf{M_S}$ of sent messages and if the deviation contains more than just errors. In the $\mathsf{aINT\text{-}CST}$ experiment the adversary wins if the string C_R submitted (in an arbitrary fragmented way) for decryption deviates from the concatenation $\|\mathbf{C_S}$ of ciphertexts that have been sent and if such deviation causes aRecv to output valid (genuine or non-genuine) messages. In different words, ciphertext-stream integrity is violated if aRecv produces any valid message once it lost synchronization. It is worth mentioning that, as for confidentiality, the exact point where synchronization is lost is defined to align with one of the (sent) ciphertext boundaries and, in contrast to the streaming setting, does not coincide with the first deviating bit of ciphertext. To detect where this point falls the $\mathcal{O}_{\mathsf{Recv}}$ oracle uses the same mechanism as in the $\mathsf{aIND\text{-}CCFA}$ experiment (see lines 25ff. of Figure 11).

Definition 6.4 ($\mathsf{aINT\text{-}PTXT}$ and $\mathsf{aINT\text{-}CST}$ Security)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel supporting fragmentation and* $\mathsf{Expt}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aINT\text{-}atk}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 11, where* atk *is a placeholder for either* PTXT *or* CST*. Within the experiment, the adversary* $\mathcal{A}$ *is given access to a sending oracle* $\mathcal{O}_{\mathsf{Send}}$ *and a receiving oracle* $\mathcal{O}_{\mathsf{Recv}}$*. We say* aCh *provides* atomic-message integrity of plaintexts*, respectively,* ciphertext streams *(*$\mathsf{aINT\text{-}PTXT}$ *resp.* $\mathsf{aINT\text{-}CST}$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathsf{Adv}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aINT\text{-}atk}}(\lambda) := \Pr\left[\mathsf{Expt}_{\mathsf{aCh},\mathcal{A}}^{\mathsf{aINT\text{-}atk}}(1^\lambda) = 1\right].$$

```
$\mathsf{Expt}^{\mathsf{aINT\text{-}atk}}_{\mathsf{aCh},\mathcal{A}}(1^\lambda)$:
$(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{aInit}(1^\lambda)$
$\mathsf{sync} \leftarrow 1$, $\mathsf{win} \leftarrow 0$
$i \leftarrow 0$, $j \leftarrow 1$
$\mathbf{M_S} \leftarrow ()$, $\mathbf{C_S} \leftarrow ()$
$\mathbf{M_R} \leftarrow ()$, $C_R \leftarrow \varepsilon$
$\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot), \mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
return $\mathsf{win}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m)$:
$(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_S, m)$
$i \leftarrow i + 1$
$\mathbf{M_S}[i] \leftarrow m$
$\mathbf{C_S}[i] \leftarrow c$
return $c$ to $\mathcal{A}$

aINT-PTXT
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
$(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
$\mathbf{M_R} \leftarrow \mathbf{M_R} \parallel \mathbf{m}$
if $\mathbf{M_R} \,\%\, [\mathbf{M_R}, \mathbf{M_S}] \notin \mathcal{E}^*$ then
    $\mathsf{win} \leftarrow 1$
return $\mathbf{m}$ to $\mathcal{A}$

aINT-CST
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
$(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
$C_R \leftarrow C_R \parallel c$
$\mathbf{M_R} \leftarrow \mathbf{M_R} \parallel \mathbf{m}$
if $\mathsf{sync} = 0$ then // already out-of-sync
    if $\mathbf{m} \notin \mathcal{E}^*$ then
        $\mathsf{win} \leftarrow 1$
else if $C_R \not\preccurlyeq \parallel \mathbf{C_S}$ then // deviating or exceeding
    while $j \leq i$ and $\parallel \mathbf{C_S}[1, \ldots, j] \preccurlyeq C_R$ and $\mathbf{M_S}[1, \ldots, j] \preccurlyeq \mathbf{M_R}$
        do $j \leftarrow j + 1$
    if $j \leq i$ or $|\mathbf{M_R}| > i$ then
        // deviation, or exceeding portion produces output
        $\mathsf{sync} \leftarrow 0$
    if $j > |\mathbf{M_R}|$ then
        $\mathbf{m}' \leftarrow ()$
    else
        $\mathbf{m}' \leftarrow \mathbf{M_R}[j, \ldots, |\mathbf{M_R}|]$
    if $\mathbf{m}' \notin \mathcal{E}^*$ then
        $\mathsf{win} \leftarrow 1$
return $\mathbf{m}$ to $\mathcal{A}$
```

Figure 11: Security experiment for *integrity* (aINT-atk with atk $\in$ {PTXT, CST}) of atomic-message channels. A PTXT-attacker is provided with access to the left $\mathcal{O}_{\mathsf{Recv}}$ oracle (aINT-PTXT), whereas a CST-attacker is instead granted access to the oracle on the right-hand side (aINT-CST).

6.3 Relations Amongst Notions

Here we explore how the different security notions for atomic-message channels supporting fragmentation relate to each other. Similarly to the case of stream-based channels, we can show that integrity of ciphertext streams implies integrity of plaintext streams, that indistinguishability against chosen-ciphertext attacks implies indistinguishability against chosen-plaintext attacks, and that the weaker confidentiality notion together with ciphertext integrity and (an adaptation of) error predictability imply the stronger confidentiality notion.

The relation aIND-CCFA $\implies$ aIND-CPA immediately follows from the fact that the aIND-CCFA experiment gives access to a left-or-right oracle as in the aIND-CPA experiment as well as to a decryption oracle. In particular, a successful chosen-plaintext attack can be seen as a successful chosen ciphertext-fragment attack that ignores the decryption oracle.

To see that aINT-CST $\implies$ aINT-PTXT it suffices to observe that, in order to produce a deviation in the sequence of messages accepted by aRecv an adversary must submit for decryption a sequence of ciphertext fragments that, in turn, deviates from the genuine ciphertext sequence so that the deviating part produces some valid messages. Put differently, an adversary cannot violate the plaintext integrity property without violating integrity of ciphertext streams. We formalize this intuition and provide a formal proof in Appendix D.

As in the case of stream-based channels, we can prove that confidentiality against passive adversaries in combination with ciphertext-stream integrity can be lifted to confidentiality against active adversaries by additionally requiring that decryption errors are efficiently predictable. For this we adapt our stream-based error predictability notion (see Definition 4.4) to the atomic-message setting in the natural way by

```
$\mathsf{Expt}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh},\mathsf{Pred},\mathcal{A}}(1^\lambda)$:
  $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{aInit}(1^\lambda)$
  $\mathsf{win} \leftarrow 0,\ i \leftarrow 0$
  $\mathbf{C_S} \leftarrow ()$
  $C_R \leftarrow \varepsilon$
  $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot), \mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
  return $\mathsf{win}$
```

```
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m)$:
  $(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_S, m)$
  $i \leftarrow i + 1$
  $\mathbf{C_S}[i] \leftarrow c$
  return $c$ to $\mathcal{A}$
```

```
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
  $(\mathsf{st}_R, \mathbf{m}) \leftarrow \mathsf{aRecv}(\mathsf{st}_R, c)$
  if $\langle \mathbf{m} \rangle_{\mathcal{E}} \neq \mathsf{Pred}(\mathbf{C_S}, C_R, c)$ then
    $\mathsf{win} \leftarrow 1$
  $C_R \leftarrow C_R \parallel c$
  return $\mathbf{m}$ to $\mathcal{A}$
```

Figure 12: Security experiment for *error predictability* ($\mathsf{aERR\text{-}PRE}$) of atomic-message channels. We denote by $\langle \cdot \rangle_{\mathcal{E}} \colon (\mathcal{M} \cup \mathcal{E})^* \to \mathcal{E}^*$ the 'projection on the error space', i.e., the mapping that removes from a vector all occurrences that do not belong to the error space $\mathcal{E}$. For instance, if $\mathbf{m} = (m_1, \perp_1, m_2, m_3, \perp_2)$ with $m_1, m_2, m_3 \in \mathcal{M}$ then $\langle \mathbf{m} \rangle_{\mathcal{E}} = (\perp_1, \perp_2)$.

demanding the predictor to output the vector of errors that aRecv would return on input a given sequence of ciphertext fragments. More formally, we say that an atomic-message channel provides (atomic-message) *error predictability* ($\mathsf{aERR\text{-}PRE}$) with respect to an efficient probabilistic *predictor* algorithm Pred if this predictor Pred, given the vector $\mathbf{C_S}$ of ciphertexts sent, the string C_R of ciphertext fragments received-so-far, and the 'next' ciphertext fragment c, returns a (potentially empty) vector containing all the errors that aRecv would output on input the (arbitrarily fragmented) string $C_R \parallel c$, in the same order they appear when output by the latter.

Definition 6.5 (Atomic-message error predictability ($\mathsf{aERR\text{-}PRE}$)). *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be an atomic-message channel with message space* $\mathcal{M}$ *and error space* $\mathcal{E}$*, and let* Pred *be an efficient probabilistic algorithm. We say that* aCh *provides* (atomic-message) error predictability ($\mathsf{aERR\text{-}PRE}$) with respect to Pred *if for every PPT adversary* $\mathcal{A}$ *playing the experiment* $\mathsf{aERR\text{-}PRE}$ *defined in Figure 12 against channel* aCh*, the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh},\mathsf{Pred},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh},\mathsf{Pred},\mathcal{A}}(1^\lambda) = 1\right].$$

We are now ready to state a composition result for atomic-message channels analogous to that for stream-based channels.

Theorem 6.6 ($\mathsf{aINT\text{-}CST} \wedge \mathsf{aIND\text{-}CPA} \wedge \mathsf{aERR\text{-}PRE} \Longrightarrow \mathsf{aIND\text{-}CCFA}$). *Let* aCh *be a (correct) atomic-message channel. If* aCh *provides integrity of ciphertext streams (*$\mathsf{aINT\text{-}CST}$*), indistinguishability against chosen-plaintext attacks (*$\mathsf{aIND\text{-}CPA}$*), as well as error predictability (*$\mathsf{aERR\text{-}PRE}$*) with respect to a predictor* Pred*, then it also provides indistinguishability against chosen ciphertext-fragment attacks (*$\mathsf{aIND\text{-}CCFA}$*).*

To prove this relation we can apply essentially the same strategy used in the proof of Theorem 4.5 and, for this reason, we abstain from providing a full proof but recall the informal argument. Assume that we have an adversary $\mathcal{A}$ attacking the $\mathsf{aIND\text{-}CCFA}$ property of a channel that provides $\mathsf{aIND\text{-}CPA}$, $\mathsf{aINT\text{-}CST}$, and $\mathsf{aERR\text{-}PRE}$. Then given only CPA capabilities one can answer $\mathcal{A}$'s queries by forwarding sending queries to the left-or-right oracle provided by the $\mathsf{aIND\text{-}CPA}$ experiment, returning empty vectors in response to in-sync decryption queries, and returning the output of the error predictor on input out-of-sync decryption queries. The $\mathsf{aINT\text{-}CST}$ property ensures that no valid message originates from out-of-sync decryption queries, while the $\mathsf{aERR\text{-}PRE}$ property allows to (efficiently) compute decryption errors.

7 Generic Construction of Atomic-Message Channels from Stream-Based Channels

In practice, applications relying on atomic-message processing perform some encoding of those messages prior to handing them over to the underlying (stream-based) secure channel. The HTTP protocol provides two prime examples for such encoding approaches. HTTP headers are encoded by having an empty line indicate the end of the header section [FR14, Section 3], i.e., a header message ends with the distinguished "end-of-message" marker `"\n\n"` which is not allowed to occur anywhere else in the header.[14] An HTTP body message in contrast can be an arbitrary byte string (i.e., the specification cannot single out a distinguished end-of-message symbol) and is hence, as one option, demarcated through indicating the body length in the `Content-Length` header field [FR14, Section 3.3]. Of course there are numerous alternative approaches to encode atomic messages in a bit stream; prepending the message with a fixed-length binary encoding of its length is a particularly efficient one, applicable whenever (an upper bound on) the message length is known.

For our generic construction that enables secure transmission of atomic messages over a generic secure stream-based channel we capture all these and further approaches under the framework of instantaneously decodable encodings.

7.1 Length-Regular Instantaneously Decodable Encoding Schemes

We recall the properties of *length-regular instantaneously decodable encoding schemes* that will be later used as a tool in our construction. The idea of using instantaneously decodable encodings was already employed by Boldyreva et al. [BDPS12] in the context of symmetric encryption supporting fragmentation in order to encode (atomic) messages in ciphertexts that might be fragmented. Such an encoding consists of an algorithm Encode that turns a word w into a codeword v, and an algorithm Decode which takes a string as input and outputs a vector $\mathbf{w}$ of words and a string s (the latter is, in fact, the part of the input string which contains no full words as a prefix).

Definition 7.1 (Length-regular instantaneously decodable encoding schemes)**.** *An* encoding scheme *with word space $W \subseteq \{0,1\}^*$ is a pair $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ of efficient deterministic algorithms defined as follows. The encoding algorithm Encode takes as input a word $w \in W$ and returns a codeword $v \leftarrow \mathsf{Encode}(w)$ where $v \in \{0,1\}^*$. The decoding algorithm Decode takes as input a string $v' \in \{0,1\}^*$ and outputs a (potentially empty) vector of words $\mathbf{w}' \in W^*$ and a string $s' \in \{0,1\}^*$. We indicate this by writing $(\mathbf{w}', s') \leftarrow \mathsf{Decode}(v')$. We use the shorthand $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ to indicate that Encode is executed sequentially on the components of $\mathbf{w} = (w_1, \ldots, w_n) \in W^*$ and the corresponding codewords are the components of $\mathbf{v} = (v_1, \ldots, v_n) \in (\{0,1\}^*)^*$ with $v_i \leftarrow \mathsf{Encode}(w_i)$.*

We say that ES is instantaneously decodable *if for all $\mathbf{w} \in W^*$ and $s \in \{0,1\}^*$, and for $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ and $(\mathbf{w}', s') \leftarrow \mathsf{Decode}(v_1 \parallel \ldots \parallel v_n \parallel s)$ where $\mathbf{v} = (v_1, \ldots, v_n)$, the following two properties hold:*

ID1. $\mathbf{w} \preccurlyeq \mathbf{w}'$, i.e., all input words from $\mathbf{w}$ are recovered by Decode in $\mathbf{w}'$ (and potentially further words contained in the string s), and

ID2. If there is no $w \in W$ such that $\mathsf{Encode}(w) \preccurlyeq s$ then $\mathbf{w}' = \mathbf{w}$ and $s' = s$, i.e., if s does not contain an encoded word then Decode recovers exactly the words in $\mathbf{w} = \mathbf{w}'$ and puts the remaining bits in s'.

We furthermore say that ES is length-regular *if for all $\mathbf{w}'$, $\mathbf{w}$ with $|\mathbf{w}'| = |\mathbf{w}|$ it holds that $|\mathbf{v}| = |\mathbf{v}'|$ where $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ and $\mathbf{v}' \leftarrow \mathsf{Encode}(\mathbf{w}')$.*

Since in this paper we only make use of encoding schemes that are instantaneously decodable and length-regular, from now on the term 'encoding scheme' refers to schemes fulfilling these properties.

Remark 7.2. For every $\mathbf{w} \in W^*$ and $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{w})$ it holds that $\mathsf{Decode}(\parallel \mathbf{v}) = (\mathbf{w}, \varepsilon)$. Indeed, for $\mathbf{v} = (v_1, \ldots, v_n)$, $s = \varepsilon$ and $(\mathbf{w}', s') \leftarrow \mathsf{Decode}(v_1 \parallel \cdots \parallel v_n \parallel s)$ property (ID2) implies $\mathbf{w}' = \mathbf{w}$ and $s' = \varepsilon$.

[14] As a technical side remark, violating the HTTP header decoding rules was part of what enabled the 'cookie-cutter' attack [BDF+14].

Remark 7.3. The set of codewords $V = \{\mathsf{Encode}(w) \mid w \in W\}$ induced by ES is *prefix-free*. Indeed, let $v, v' \in V$ be such that $v \preccurlyeq v'$ and write $v' = v \parallel s$ for some $s \in \{0,1\}^*$. By definition there exist $w, w' \in W$ such that $v = \mathsf{Encode}(w)$ and $v' = \mathsf{Encode}(w')$. By Remark 7.2 we have $\mathsf{Decode}(v') = ((w'), \varepsilon)$; similarly, by property (ID2) we derive $\mathsf{Decode}(v') = \mathsf{Decode}(v \parallel s) = (\mathbf{w}'', s'')$ where $(w) \preccurlyeq \mathbf{w}''$. Putting these relations together we get $(w) \preccurlyeq \mathbf{w}'' = (w') \Longrightarrow w = w'$ and thus $v = \mathsf{Encode}(w) = \mathsf{Encode}(w') = v'$. In what follows, when writing that an encoding scheme is prefix-free we mean that its set of codewords is.

Example 7.4 (The end-of-message encoding)**.** Take any string $\diamond \in \{0,1\}^*$ and let $\kappa = |\diamond|$ be its length. Define $W \subset \{0,1\}^*$ recursively in such a way that no (finite) concatenation of words in W contains the distinguished string $\diamond$. Formally, we require that for all $u \in W$ with $|u| \geq \kappa$ and for all i such that $1 \leq i \leq |u| - \kappa$ it holds $\diamond \neq u[i, \ldots, i+\kappa]$. We define the *end-of-message encoding* through the following functions. We encode $m \in W$ by appending to it the end-of-message symbol $\diamond$, i.e., $\mathsf{Encode}(m) = m \parallel \diamond$. To decode a string $y \in \{0,1\}^*$ we first initialize $\mathbf{w} \leftarrow ()$, $s \leftarrow \varepsilon$, then we scan y from left to right until we find the first occurrence of $\diamond$. If there is none, we set $s = y$ and return $(\mathbf{w}, s)$. Otherwise we found the first word w_1 that y encodes, i.e., such that $w_1 \parallel \diamond \preccurlyeq y$; thus, we append w_1 to $\mathbf{w}$ and proceed (recursively) as above using the unprocessed string $y' \leftarrow y \,\%\, (w_1 \parallel \diamond)$ instead of y. Observe that, by definition of W, any string $y \in \{0,1\}^*$ admits a unique decomposition $y = w_1 \parallel \diamond \parallel \cdots \parallel w_\ell \parallel \diamond \parallel w_{\ell+1}$ with $\ell \geq 0$, $w_1, \ldots, w_\ell \in W$ and $w_{\ell+1} \in \{0,1\}^* \setminus W$. By the obvious correctness of this algorithm, wee see that the end-of-message encoding is instantaneously decodable. It is also length-regular due to the fixed length κ of the appended end-of-message symbol $\diamond$.

A specific instance of the end-of-message encoding is the HTTP header encoding where $\diamond =$ `\n\n` (i.e., two ASCII newline characters) and header messages follow a specific format which in particular forbids two subsequent newlines to occur in a message.

7.2 The Encode-then-Stream Construction

For our generic construction of an atomic-message channel from a stream-based channel we now leverage a length-regular instantaneously decodable encoding scheme $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ with word space $W = \mathcal{M}$ as a generalization of both real-world and theoretical approaches to convert atomic messages into a stream and, vice versa, a stream into (a vector of) atomic messages. We name this paradigm *encode-then-stream* and denote by $\mathsf{aCh}_{\mathsf{EtS}}$ the resulting atomic-message channel.

The main idea is very natural: we encode atomic messages within the message stream sent and, on the receiver's side, to buffer the incoming stream and only output messages once they are received completely. Briefly, algorithm aSend takes an atomic message m and first encodes it by invoking $v \leftarrow \mathsf{Encode}(m)$. It then processes the corresponding string v using the stream-based channel's sending algorithm (with a flush request), obtaining $(\mathsf{st}_S, c) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_S, v, 1)$. Correspondingly, algorithm aRecv takes as input a ciphertext fragment $c \in \{0,1\}^*$ and first invokes the streaming receiving algorithm $(\mathsf{st}_R, v) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$. It then extracts from v the longest prefix v' that does not contain error symbols and concatenates the latter to the buffer, $\mathsf{buf} \leftarrow \mathsf{buf} \parallel v'$. Finally, it decodes the new buffer content to obtain an atomic-message vector and an updated (potentially empty) buffer, $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$.

Construction 7.5 (Encode-then-stream construction $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Consider a stream-based channel* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with error space* $\mathcal{E}$ *and an encoding scheme* $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ *with word space* $W \subseteq \{0,1\}^*$*. We define* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *to be the atomic-message channel with message space* $\mathcal{M} = W$ *and error space* $\{\perp\}$ *obtained by applying to* Ch *the transform described in Figure 13.*

Correctness of $\mathsf{aCh}_{\mathsf{EtS}}$ directly follows from the correctness of Ch and the instantaneous decodability of ES, as we show in the following proposition.

```
$\mathsf{aInit}(1^\lambda)$:
    $(\mathsf{st}'_{S,0}, \mathsf{st}'_{R,0}) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
    $\mathsf{buf} \leftarrow \varepsilon$
    $\mathsf{fail} \leftarrow 0$
    $\mathsf{st}_{S,0} = \mathsf{st}'_{S,0}$
    $\mathsf{st}_{R,0} = (\mathsf{st}'_{R,0}, \mathsf{buf}, \mathsf{fail})$
    return $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$
```

```
$\mathsf{aSend}(\mathsf{st}_S, m)$:
    $v \leftarrow \mathsf{Encode}(m)$
    $(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, v, 1)$
    return $(\mathsf{st}_S, c)$
```

```
$\mathsf{aRecv}(\mathsf{st}_R, c)$:
    parse $\mathsf{st}_R$ as $(\mathsf{st}'_R, \mathsf{buf}, \mathsf{fail})$
    if $\mathsf{fail} = 1$ then
        return $(\mathsf{st}_R, (\bot))$
    $(\mathsf{st}'_R, v) \leftarrow \mathsf{Recv}(\mathsf{st}'_R, c)$
    $\ell = \max\{|u| : u \preccurlyeq v \wedge u \in \{0,1\}^*\}$
    $v' \leftarrow v[1, \ldots, \ell]$ // longest non-error prefix of $v$
    $\mathsf{buf} \leftarrow \mathsf{buf} \parallel v'$
    $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$
    if $v' \neq v$ then
        $\mathsf{fail} \leftarrow 1$
        $\mathbf{m} \leftarrow \mathbf{m} \parallel (\bot)$
    $\mathsf{st}_R \leftarrow (\mathsf{st}'_R, \mathsf{buf}, \mathsf{fail})$
    return $(\mathsf{st}_R, \mathbf{m})$
```

Figure 13: Generic construction of an atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ with message space $\mathcal{M}$ from any stream-based channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ and an encoding scheme $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ with word space $W = \mathcal{M}$.

Proposition 7.6 (Correctness of $\mathsf{aCh}_{\mathsf{EtS}}$). *If the stream-based channel* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *is correct and the encoding scheme* $\mathsf{ES} = (\mathsf{Encode}, \mathsf{Decode})$ *is instantaneously decodable then the atomic-message channel* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *is correct, too.*

Proof. Let $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow_\$ \mathsf{aInit}(1^\lambda)$, $\ell \in \mathbb{N}$, and $\mathbf{m} \in \mathcal{M}^\ell$ be arbitrary and let $\mathbf{c} \in (\{0,1\}^*)^\ell$ be such that $(\mathsf{st}_{S,\ell}, \mathbf{c}) \leftarrow_\$ \mathsf{aSend}(\mathsf{st}_{S,0}, \mathbf{m})$. Let $\ell' \in \mathbb{N}$ and $\mathbf{c}' \in (\{0,1\}^*)^{\ell'}$ be arbitrary, and let $\mathbf{m}' \in \mathcal{M}^{\ell'}$ be such that $(\mathsf{st}_{R,\ell'}, \mathbf{m}') \leftarrow \mathsf{aRecv}(\mathsf{st}_{R,0}, \mathbf{c}')$. Suppose that for some $i \in [1, \ldots, \ell]$ it holds $\| \mathbf{c}[1 \ldots i] \preccurlyeq \| \mathbf{c}' \preccurlyeq \mathbf{c}$. Using a similar notation let $\mathbf{v} \in (\{0,1\}^*)^\ell$ denote the vector of codewords $\mathbf{v} \leftarrow \mathsf{Encode}(\mathbf{m})$ and let $\mathbf{v}' \in (\{0,1\}^*)^{\ell'}$ be such that $(\mathsf{st}'_{R,\ell'}, \mathbf{v}') \leftarrow \mathsf{Recv}(\mathsf{st}'_{R,0}, \mathbf{c}')$. Observe that $\mathbf{c}$ is generated by invoking the stream-based sending algorithm Send on input the components of $\mathbf{v}$ and flush flags $\mathbf{f} = (1, \ldots, 1)$. Then, by (stream-based) correctness of Ch we have that for all $j \in [1, \ldots, \ell]$, and in particular for $j = i$, it holds $\| \mathbf{c}[1, \ldots, j] \preccurlyeq \| \mathbf{c}' \preccurlyeq \| \mathbf{c} \implies \| \mathbf{v}[1, \ldots, j] \preccurlyeq \| \mathbf{v}' \preccurlyeq \| \mathbf{v}$. By construction we now have $(\mathbf{m}[1, \ldots, i], \varepsilon) \leftarrow \mathsf{Decode}(\| \mathbf{v}[1, \ldots, i])$, $(\mathbf{m}', s) \leftarrow \mathsf{Decode}(\| \mathbf{v}')$ for some $s \in \{0,1\}^*$, and $(\mathbf{m}, \varepsilon) \leftarrow \mathsf{Decode}(\| \mathbf{v})$. Using the property ID1 of encoding schemes we derive from $\| \mathbf{v}[1, \ldots, i] \preccurlyeq \| \mathbf{v}'$ that $\mathbf{m}[1, \ldots, i] \preccurlyeq \mathbf{m}'$. Furthermore, $\mathbf{m}' \preccurlyeq \mathbf{m}$ as property ID2 ensures that Decode only decodes full codewords in $\| \mathbf{v}' \,\%\, \| \mathbf{v}[1, \ldots, i]$ namely, by property ID1 and as $\| \mathbf{v}' \preccurlyeq \| \mathbf{v}$, those codewords $\mathbf{v}[i+1, \ldots, \ell]$ corresponding to $\mathbf{m}[i+1, \ldots, \ell]$ fully contained in $\| \mathbf{v}'$. $\square$

7.3 Security of Encode-then-Stream

Ideally we would like to show that confidentiality and integrity properties of the stream-based channel Ch can be lifted to the corresponding security properties of the atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$. Indeed, using a specific encoding to identify message boundaries—as our generic construction $\mathsf{aCh}_{\mathsf{EtS}}$ does—is a natural approach to encode atomic messages within a stream of bits. This approach is pursued by numerous application layer protocols including, among others, HTTP [FR14]. Surprisingly, not even the strongest confidentiality and integrity properties of the underlying stream-based channel Ch (i.e., $\mathsf{IND\text{-}CCFA}$ and $\mathsf{INT\text{-}CST}$) suffice to make our atomic-message channel construction $\mathsf{aCh}_{\mathsf{EtS}}$ $\mathsf{aIND\text{-}CCFA}$- and $\mathsf{aINT\text{-}CST}$-secure.

As a particular (and admittedly artificial) counterexample consider the following variant $\mathsf{Ch}'_{\mathsf{AEAD}} = (\mathsf{Init}', \mathsf{Send}', \mathsf{Recv}')$ of the AEAD-based streaming channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ (see Construction 5.1): The Send' algorithm processes the input message as Send does, but additionally appends a 0-bit to each AEAD ciphertext c' computed by Send. On the receiver's side, the AEAD ciphertext is processed as before and, if the appended bit following the AEAD ciphertext (which is also allowed to arrive in a separate fragment) equals 1, then after decrypting the ciphertext and adding the result to the output message m the failure flag is set to $\mathsf{fail} \leftarrow 1$ and the error symbol $\perp$ is appended to m.

Observe that construction $\mathsf{Ch}'_{\mathsf{AEAD}}$ preserves the IND-CCFA and INT-CST security properties of the original stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}}$ (see Section 5) since, intuitively, the ability to flip the redundant bit allows the adversary only to create an extra error symbol which harms neither confidentiality nor integrity. The reason is that the message part from the unaltered ciphertext prefix, without the extra bit, is still suppressed in the stream-based confidentiality experiment. Analogously, creating additional error symbols in the stream-based integrity experiment does not violate security. However, as we show next, using $\mathsf{Ch}'_{\mathsf{AEAD}}$ as the underlying stream-based channel results in the atomic-message construction $\mathsf{aCh}_{\mathsf{EtS}}$ being insecure with respect to both confidentiality and integrity, as we discuss next.

Let us consider confidentiality first: for any ciphertext $c = len \parallel c' \parallel 0$ (where $len \parallel c'$ is the ciphertext as generated by $\mathsf{Ch}_{\mathsf{AEAD}}$) output by the left-or-right oracle, an adversary against the aIND-CCFA security of $\mathsf{aCh}_{\mathsf{EtS}}$ can simply query the receiving oracle on $c^* = len \parallel c' \parallel 1$ and will obtain the input message m'_b used on the sender's side. Indeed, as the AEAD ciphertext c' is unmodified, the AEAD decryption will yield the full codeword $v' = \mathsf{Encode}(m'_b)$. Therefore the (atomic-message) receiving oracle $\mathcal{O}_{\mathsf{Recv}}$, treating c^* as an atomic (and hence differing) ciphertext, will return the pair $(m'_b, \perp)$ to the adversary, allowing it to break confidentiality. However, since in the streaming setting the message m'_b is already output upon receiving the genuine part $\widetilde{c^*} = len \parallel c'$ of the ciphertext c^*, stream-based confidentiality of $\mathsf{Ch}'_{\mathsf{AEAD}}$ is not affected by this attack.

A similar argument applies to the case of integrity: an adversary against the aINT-CST property of $\mathsf{aCh}_{\mathsf{EtS}}$ can, given a ciphertext $c = len \parallel c' \parallel 0$, flip the last bit of c and make the receiving oracle rate the resulting message m' (where $m' = \mathsf{Decode}(v')$ and v' is the codeword obtained by AEAD decrypting c') as a valid message output on a deviating ciphertext c^*. The latter breaks the atomic ciphertext integrity of $\mathsf{aCh}_{\mathsf{EtS}}$ without violating the notion of stream-based integrity of $\mathsf{Ch}'_{\mathsf{AEAD}}$.

We stress that the modified construction $\mathsf{Ch}'_{\mathsf{AEAD}}$ certainly constitutes a particularly unnatural, yet secure stream-based channel. It nevertheless indicates the need for an additional requirement on the stream-based channel, ruling out such behavior, for proving the security of $\mathsf{aCh}_{\mathsf{EtS}}$ generically.[15] We conjecture that this is not a limitation specific to our $\mathsf{aCh}_{\mathsf{EtS}}$ construction but that indeed no generic atomic-message channel construction working, from a protocol-layering perspective, on top of $\mathsf{Ch}'_{\mathsf{AEAD}}$ in a black-box manner can satisfy confidentiality or integrity. As an attempt to formalize the additional requirement just mentioned, we propose a new security notion that precludes a stream-based channel from behaving like $\mathsf{Ch}'_{\mathsf{AEAD}}$ and prove it sufficient, together with INT-CST and IND-CCFA security, for the security of the encode-then-stream paradigm.

7.3.1 Conciseness of ciphertext streams

Intuitively, we want to ensure that if one submits a *strict prefix* of the genuine ciphertext stream for decryption then the *complete* message stream will not be output on the receiver's side. This, in particular, rules out those constructions for which Send appends redundant bits to the ciphertext fragment (as the construction $\mathsf{Ch}'_{\mathsf{AEAD}}$ in our example above) that can be chopped without affecting the underlying message

[15]We note that Canetti et al. [CKN03] introduced a relaxed notion of confidentiality (for public-key encryption, but also applicable to the secret-key setting), so-called indistinguishability under replayable chosen-ciphertext attacks (RCCA), capturing that an encryption scheme is non-malleable beyond trivial ciphertext modifications (like our additional, flippable bit). It is plausible that the channel $\mathsf{Ch}_{\mathsf{AEAD}}$ achieves a similar, relaxed notion of confidentiality for stream-based channels (an RCCA-style notion in the streaming setting is yet to be defined, though). Here we take a different route and, aiming at a stronger, CCA-like confidentiality property, we rather make explicit an additional property for stream-based channels—conciseness of ciphertexts—that enables lifting confidentiality and integrity of the underlying stream-based channel to the constructed atomic-message channel.

```
$\mathsf{Expt}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$:
1  $(\mathsf{st}_S, \mathsf{st}_R) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
2  $i \leftarrow 0, j \leftarrow 1$
3  $\mathsf{win} \leftarrow 0$
4  $\mathbf{M_S} \leftarrow (), \mathbf{C_S} \leftarrow ()$
5  $M_R \leftarrow \varepsilon, C_R \leftarrow \varepsilon$
6  $\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
7  if $C_R \prec \| \mathbf{C_S}$ then
8    while $\| \mathbf{C_S}[1,\ldots,j] \preccurlyeq C_R$
9      do $j \leftarrow j+1$
       // $\mathbf{C_S}[j]$ is first ciphertext not received completely
10   if $\| \mathbf{M_S}[1,\ldots,j] \preccurlyeq M_R$ then
11     $\mathsf{win} \leftarrow 1$
12 return $\mathsf{win}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m,f)$:
13 $(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, m, f)$
14 $i \leftarrow i+1$
15 $\mathbf{M_S}[i] \leftarrow m$
16 $\mathbf{C_S}[i] \leftarrow c$
17 return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
18 $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
19 $M_R \leftarrow M_R \| m$
20 $C_R \leftarrow C_R \| c$
21 return $m$ to $\mathcal{A}$
```

Figure 14: Security experiment for *conciseness of ciphertext streams* (CON-CST) for stream-based channels.

fragment. Put differently, we require that accepted ciphertexts are concise. We thus name this new security property *conciseness of ciphertext streams* (CON-CST).[16] The intuition behind our definition is as follows. The adversary's goal is to deliver only a strict prefix C_R of the sender's output stream $\| \mathbf{C_S}$ of the atomic ciphertexts to the receiver, but such that the receiver still obtains all message chunks. In other words, the adversary wins if it manages to cut some bits in the ciphertext stream (such as a redundant bit in a ciphertext) without affecting the message delivery. As we will see below, conciseness is naturally achievable by stream-based channel constructions, including ours. Investigating the necessity of conciseness (or a different notion) for sending atomic messages over a stream-based channel in a protocol-layered manner is a possible avenue for future work.

Definition 7.7 (Conciseness of ciphertext streams (CON-CST)). *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 14. Within the experiment, the adversary* $\mathcal{A}$ *is given access to a sending oracle* $\mathcal{O}_{\mathsf{Send}}$ *and a receiving oracle* $\mathcal{O}_{\mathsf{Recv}}$*. We say* Ch *provides* conciseness of ciphertext streams *(*CON-CST*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible:*

$$\mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right].$$

Remark 7.8. It is easy to see that for a stream-based channel with concise ciphertext streams, the Send algorithm can never output a non-empty ciphertext $c \neq \varepsilon$ on input an empty message $m = \varepsilon$ and having no buffered message input from previous calls. Assume otherwise, i.e., a sequence of message fragments $\mathbf{M_S}[1], \ldots, \mathbf{M_S}[i-1], \mathbf{M_S}[i]$ transformed by Send (with the flush flag always set to $f = 1$) into a sequence of ciphertext fragments $\mathbf{C_S}[1], \ldots, \mathbf{C_S}[i-1], \mathbf{C_S}[i]$; with $\mathbf{M_S}[i] = \varepsilon$ while $\mathbf{C_S}[i] \neq \varepsilon$. Now, an adversary can simply query $\mathcal{O}_{\mathsf{Recv}}$ on $\mathbf{C_S}[1] \| \ldots \| \mathbf{C_S}[i-1]$ first, next query $\mathcal{O}_{\mathsf{Recv}}$ on the bit-wise inverse of $\mathbf{C_S}[i]$, and then stop. It wins the CON-CST experiment, as $\mathbf{C_S}[i]$ is not contained in the received ciphertext stream C_R, yet all message fragments including $\mathbf{M_S}[i]$ are contained in the received message stream M_R, as $\mathbf{M_S}[1] \| \ldots \| \mathbf{M_S}[i-1] \| \mathbf{M_S}[i] = \mathbf{M_S}[1] \| \ldots \| \mathbf{M_S}[i-1]$ (recall that $\mathbf{M_S}[i] = \varepsilon$) and the latter is contained by correctness.

[16]The scope of conciseness for stream-based channels is somewhat similar in spirit to that of *tidiness* [NRS14] for nonce-based encryption: it is a natural requirement to rule out schemes for which the receiving algorithm performs some useless operation allowing an adversary to trivially break security.

We remark that in practice channel protocols including IPsec and TLS do specify the possibility to send empty message fragments as a traffic analysis countermeasure; yet, popular libraries (e.g., OpenSSL [Ope]) do not allow this option. The common 'encode-then-stream' approach which we capture in our generic $\mathsf{aCh}_{\mathsf{EtS}}$ construction does not rely on the capability of sending empty message fragments. We hence do consider the restriction to disallow empty message fragments as non-critical in this setting.

7.3.2 Integrity and Confidentiality of Encode-then-Stream

We now turn towards analyzing the security of our generic construction of an atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$ from a stream-based channel Ch and an encoding scheme ES. In brief, we prove that integrity of ciphertext streams (INT-CST) of the stream-based channel Ch can be lifted to the analogous property (aINT-CST) for the resulting atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$, provided that Ch also offers conciseness of ciphertexts (CON-CST). Due to the subtle difference between the synchronization mechanisms in the streaming and the atomic-message setting, the proof of this result is quite involved. We also show that indistinguishability under chosen plaintext-fragment attacks (IND-CPFA) and error predictability (ERR-PRE) of Ch can be lifted to the analogous properties (aIND-CPA and aERR-PRE) of $\mathsf{aCh}_{\mathsf{EtS}}$. These relations together with Theorem 6.6 imply that INT-CST, IND-CPFA, CON-CST and ERR-PRE of Ch are sufficient conditions for the encode-then-stream paradigm to provide indistinguishability under chosen ciphertext-fragment attacks (IND-CCFA).

It is worth noting that, while integrity of ciphertext streams (INT-CST), given CON-CST, directly carries over from the stream-based channel Ch to the atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$, the same does not seem to hold for confidentiality against active adversaries (IND-CCFA). While our proof lifts aIND-CPA to aIND-CCFA security by leveraging ciphertext-stream integrity and error predictability (via the compositional result from Theorem 6.6), one may wonder whether requiring INT-CST and ERR-PRE is indeed necessary. A different route to achieve aIND-CCFA security of the encode-then-stream paradigm could be used to lift confidentiality against an active adversary directly from the IND-CCFA security of Ch (recall that CON-CST of Ch would be necessary also in this case, as explained at the beginning of Section 7.3). At first glance, one might expect lifting IND-CCFA to its analog in the atomic-message setting, aIND-CCFA, should not rely on integrity of the stream-based channel. We however conjecture that such integrity is in fact necessary. Without going into details, the reason for this is that a non-integrous stream-based channel may possibly allow an attacker to modify the sent message stream in an arbitrary manner from some point on. In particular, the adversary might be able to modify the atomic-message encoding, e.g., by moving an employed end-of-message symbol to some earlier position in the message stream. Such a modification does not imply a stream-based confidentiality break, as the preceding challenge-message stream would still be suppressed. In the atomic-message sense, however, the resulting received (challenge) message is shortened, hence considered to be different and output to the adversary in the aIND-CCFA experiment. We therefore expect that stream-based integrity is indeed necessary to bridge the gap from stream-based IND-CCFA to atomic-message aIND-CCFA security. This, in particular, provides a glimpse into the formal causes enabling the cookie cutter attack [BDF^{+}14]: ultimately, atomic-message encodings need integrity protection as otherwise an adversary can restructure application messages (in this case application-layer HTTP messages) in a way that may not only violate their integrity, but also confidentiality.

In proving the theorem on integrity, the ideal target would be to build an efficient reduction that turns any successful aINT-CST adversary against the atomic-message channel $\mathsf{aCh}_{\mathsf{EtS}}$ into a successful INT-CST adversary against the streaming channel Ch. Intuitively, the reduction can simply perform the encoding and decoding operations of $\mathsf{aCh}_{\mathsf{EtS}}$ by itself and realize the streaming operations using sending and receiving oracles provided by the INT-CST experiment. The challenging part is to guarantee that any success in the aINT-CST game translates to a success in the INT-CST game. In fact, because of the different synchronization mechanisms adopted in the atomic-message setting and in the streaming setting, it is not possible to exploit every $\mathsf{aINT\text{-}CST}$ break against $\mathsf{aCh_{EtS}}$ to violate the $\mathsf{INT\text{-}CST}$ property of Ch. For stream-based channels, synchronization is lost starting from the first deviating bit in the ciphertext stream. In contrast, for atomic-message channels we declare the entire ciphertext to be out-of-sync as soon as a deviation in the ciphertext sequence *or* in the message sequence is detected. Therefore, the receiving oracle in the atomic-message setting may lose synchronization 'earlier' than in the streaming setting. As we will see explicitly in the proof, $\mathsf{CON\text{-}CST}$ ensures that the oracle provided to the reduction from the $\mathsf{INT\text{-}CST}$ experiment and the one that $\mathcal{A}$ is presented with from the emulated $\mathsf{aINT\text{-}CST}$ experiment are consistent.

Theorem 7.9 ($\mathsf{aINT\text{-}CST}$ security of $\mathsf{aCh_{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and let* $\mathsf{aCh_{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 7.5). If* Ch *provides integrity and conciseness of ciphertext streams (*$\mathsf{INT\text{-}CST}$ *and* $\mathsf{CON\text{-}CST}$*) then* $\mathsf{aCh_{EtS}}$ *provides atomic-message integrity of ciphertexts (*$\mathsf{aINT\text{-}CST}$*). Formally, for every efficient* $\mathsf{aINT\text{-}CST}$ *adversary* $\mathcal{A}$ *there exist an efficient* $\mathsf{CON\text{-}CST}$ *adversary* $\mathcal{B}$ *and an* $\mathsf{INT\text{-}CST}$ *adversary* $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh_{EtS}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{C}}(\lambda).$$

Proof. We will show that if $\mathcal{A}$ wins the $\mathsf{aINT\text{-}CST}$ game against $\mathsf{aCh_{EtS}}$ then we can either violate the $\mathsf{CON\text{-}CST}$ or the $\mathsf{INT\text{-}CST}$ properties of the underlying stream-based channel Ch. We start with an intuitive explanation and then give explicit reductions. Assume that $\mathcal{A}$ wins the $\mathsf{aINT\text{-}CST}$ game (from Figure 11). Then there are four possibilities for the win flag $\mathsf{win} \leftarrow 1$ to be set the first time—as we sketch below and explore in detail in the course of the proof—depending on whether $\mathsf{win} \leftarrow 1$ is set in line 23 (option #1 below) or in line 34 (options #2, #3, and #4).

#1. Synchronization has been lost before. Then C_R must be deviating from $\|\mathbf{C_S}$ and a (fully) deviating fragment causes aRecv to output some valid message.[17] As we will see, this leads to violating the $\mathsf{INT\text{-}CST}$ property of Ch.

#2. The stream C_R *goes ahead* of $\|\mathbf{C_S}$ (the loop of lines 25–26 terminates because $j > i$) and the exceeding part produces some valid message when processed by aRecv. This violates the $\mathsf{INT\text{-}CST}$ property of Ch, too.

#3. The stream C_R *deviates* from $\|\mathbf{C_S}$ after the first $j-1$ sent ciphertexts and messages are received entirely (the loop terminates because $j \leq i$ but $\|\mathbf{C_S}[1,\ldots,j] \not\preccurlyeq C_R$). Here the fact that $\mathcal{A}$ wins the $\mathsf{aINT\text{-}CST}$ game does not necessarily lead to violating the $\mathsf{INT\text{-}CST}$ property of Ch but, if not, it infringes its $\mathsf{CON\text{-}CST}$ property.

#4. The sequence $\mathbf{M_R}$ *deviates* from $\mathbf{M_S}$ after the first $j-1$ sent messages are received completely (the loop terminates because $j \leq i$ and $\|\mathbf{C_S}[1,\ldots,j] \preccurlyeq C_R$ but $\mathbf{M_S}[1,\ldots,j] \not\preccurlyeq \mathbf{M_R}[1,\ldots,j]$). Also in this case we can leverage $\mathcal{A}$'s strategy in $\mathsf{aINT\text{-}CST}$ to break the $\mathsf{INT\text{-}CST}$ security of Ch.

In the rest of the proof we first isolate the conditions causing $\mathcal{A}$ to be successful in the $\mathsf{aINT\text{-}CST}$ game without violating the $\mathsf{INT\text{-}CST}$ security of Ch—note that this can only happen for option #3. We then define a new game which penalizes $\mathcal{A}$ if the latter occurs, and bound the difference in probability between the modified game and the original one with the $\mathsf{CON\text{-}CST}$ advantage of an efficient adversary $\mathcal{B}$. Finally, we show that the modified game can be simulated by an efficient adversary $\mathcal{C}$ which breaks the $\mathsf{INT\text{-}CST}$ property whenever $\mathcal{A}$ wins the $\mathsf{aINT\text{-}CST}$ game.

[17] In principle, C_R could also be ahead of $\|\mathbf{C_S}$. However, in this case we can have $\mathsf{sync} = 0$ only if a valid message was already output upon processing some previous (ahead) fragment; but then $\mathsf{win} \leftarrow 1$ would have been already set. Note that once $\mathsf{aCh_{EtS}}$ output the first error symbol, by construction it only outputs errors from that point on. Hence, $\mathsf{win} \leftarrow 1$ cannot be set after the first error output occurred.

```
$\mathsf{E}^0_{\mathcal{A}}(1^\lambda)$, $\mathsf{E}^1_{\mathcal{A}}(1^\lambda)$:
$(\mathsf{st}'_{S,0}, \mathsf{st}'_{R,0}) \leftarrow_\$ \mathsf{Init}(1^\lambda)$
$\mathsf{buf} \leftarrow \varepsilon, \mathsf{fail} \leftarrow 0$
$\mathsf{st}_S \leftarrow \mathsf{st}'_{S,0}$
$\mathsf{st}_R \leftarrow (\mathsf{st}'_{R,0}, \mathsf{buf}, \mathsf{fail})$
$\mathsf{sync} \leftarrow 1$, $\mathsf{win} \leftarrow 0$
$i \leftarrow 0$
$\mathbf{M_S} \leftarrow ()$, $\mathbf{C_S} \leftarrow ()$
$\mathbf{M_R} \leftarrow ()$, $C_R \leftarrow \varepsilon$
$\boxed{\mathbf{V_S} \leftarrow (), V_R \leftarrow \varepsilon}$
$\mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\cdot), \mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$
$\boxed{\text{if } \mathsf{bad} = 1 \text{ then } \mathsf{win} \leftarrow 0}$
return $\mathsf{win}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Send}}(m)$:
$v \leftarrow \mathsf{Encode}(m)$
$(\mathsf{st}_S, c) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_S, v, 1)$
$i \leftarrow i + 1$
$\mathbf{M_S}[i] \leftarrow m$
$\mathbf{C_S}[i] \leftarrow c$
$\boxed{\mathbf{V_S}[i] \leftarrow v}$
return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
parse $\mathsf{st}_R$ as $(\mathsf{st}'_R, \mathsf{buf}, \mathsf{fail})$
$\boxed{\widetilde{\mathsf{st}'_R} \leftarrow \mathsf{st}'_R}$ // copy of current state
if $\mathsf{fail} = 1$ then
    $\mathbf{m} \leftarrow (\perp)$
else
    $(\mathsf{st}'_R, v) \leftarrow \mathsf{Recv}(\mathsf{st}'_R, c)$
    $\ell = \max\{|u| : u \preccurlyeq v \wedge u \in \{0,1\}^*\}$
    $v' \leftarrow v[1..\ell]$ // $v'$ is the longest error-free prefix of $v$
    $\mathsf{buf} \leftarrow \mathsf{buf} \parallel v'$
    $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$
    if $v' \neq v$ then
        $\mathsf{fail} \leftarrow 1, \mathbf{m} \leftarrow \mathbf{m} \parallel (\perp)$
    $\mathsf{st}_R \leftarrow (\mathsf{st}'_R, \mathsf{buf}, \mathsf{fail})$
$\boxed{C'_R \leftarrow C_R}$
$C_R \leftarrow C_R \parallel c$
$\mathbf{M_R} \leftarrow \mathbf{M_R} \parallel \mathbf{m}$
$\boxed{V'_R \leftarrow V_R}$
$\boxed{V_R \leftarrow V_R \parallel v}$
if $\mathsf{sync} = 0$ then
    if $\mathbf{m} \notin \mathcal{E}^*$ then
        $\mathsf{win} \leftarrow 1$
else if $C_R \not\preccurlyeq \parallel \mathbf{C_S}$ then
    $j \leftarrow 1$
    while $j \leq i$ and $\parallel \mathbf{C_S}[1, \ldots, j] \preccurlyeq C_R$
            and $\mathbf{M_S}[1, \ldots, j] \preccurlyeq \mathbf{M_R}$
        do $j \leftarrow j + 1$
    $\boxed{\text{if } \parallel \mathbf{C_S} \not\preccurlyeq C_R \text{ then}}$
        $\boxed{\widetilde{c} \leftarrow [C_R, \parallel \mathbf{C_S}] \,\%\, C'_R}$
        $\boxed{(\widetilde{\mathsf{st}'_R}, \widetilde{v}) \leftarrow \mathsf{Recv}(\widetilde{\mathsf{st}'_R}, \widetilde{c})}$
        $\boxed{\text{if } \parallel \mathbf{V_S}[1, \ldots, j] \preccurlyeq V'_R \parallel \widetilde{v} \text{ then}}$
            $\boxed{\mathsf{bad} \leftarrow 1}$
    if $j \leq i$ or $|\mathbf{M_R}| > i$ then
        $\mathsf{sync} \leftarrow 0$
    $\mathbf{m}' \leftarrow \mathbf{M_R}[j, \ldots, |\mathbf{M_R}|]$
    if $\mathbf{m}' \notin \mathcal{E}^*$ then
        $\mathsf{win} \leftarrow 1$
return $\mathbf{m}$ to $\mathcal{A}$
```

Figure 15: Security experiments $\mathsf{E}^0_{\mathcal{A}} = \mathsf{Expt}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh}_{\mathsf{EtS}}, \mathcal{A}}$ and $\mathsf{E}^1_{\mathcal{A}}$, derived from $\mathsf{E}^0_{\mathcal{A}}$ by including the $\boxed{\text{framed}}$ instructions, described in the proof of Theorem 7.9.

We first set some notation. Let E^0 denote the $\mathsf{aINT\text{-}CST}$ experiment with the algorithms of $\mathsf{aCh}_{\mathsf{EtS}}$ plugged-in, as depicted in Figure 15 (ignore the framed instructions for now). Now we define a new experiment E^1 starting from E^0 by including the framed instructions (lines 9, 11, 18, 21, 33, 36–37, and 45–49). The resulting game essentially works as E^0 but it resets $\mathsf{win} \leftarrow 0$ if, although C_R contains only up to the first $j-1$ genuine ciphertexts and then deviates from $\mathbf{C_S}$, Recv produces a stream $V'_R \parallel \widetilde{v}$ which contains the first j genuine codewords upon processing the longest genuine prefix $\widetilde{c}$ of the first deviating query c. Informally, this change isolates the event that $\mathcal{A}$ wins the $\mathsf{aINT\text{-}CST}$ experiment against $\mathsf{aCh}_{\mathsf{EtS}}$ without causing a violation of the $\mathsf{INT\text{-}CST}$ property of Ch (i.e., a deviation from $\mathbf{C_S}$ does not translate to a deviation from the underlying message stream), and prevents $\mathcal{A}$ from winning the game in such a case. In more detail, through lines 9, 18, and 36 the new game additionally maintains a sequence $\mathbf{V_S}$ for bookkeeping the codewords input to Send as well as a string V_R for the fragments output by Recv. Instruction 21 makes a copy $\widetilde{\mathsf{st}'_R}$ of the current state st'_R (of the streaming algorithm) as well as copies C'_R and V'_R of the current strings C_R and V_R before the current query is processed by Recv. Instructions 45–47 identify the first deviating query c and perform an auxiliary call to Recv (with state $\widetilde{\mathsf{st}'_R}$, i.e., prior to processing c) on the longest genuine prefix $\widetilde{c}$ of c. Finally, instructions 48–49 detect whether processing the longest genuine prefix of V_R through Recv yields the first j codewords entirely, and, if so, set the flag bad. In what follows we denote by *bad* the event that $\mathsf{bad} \leftarrow 1$ is triggered. Finally, instruction 11 penalizes the adversary if it triggers event *bad*. Since the experiments E^0 and E^1 returns the same outcome as long as *bad* does not occur, we can bound their difference in probability by

$$\left|\mathsf{Adv}^{\mathsf{E}^0}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{E}^1}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda)\right| \leq \Pr[bad].$$

We now show that the occurring of event *bad* translates to a violation of the $\mathsf{CON\text{-}CST}$ property of the stream-based channel Ch. To this end, we build an explicit reduction $\mathcal{B}$ which simulates the game for $\mathcal{A}$ using the oracles provided by the $\mathsf{CON\text{-}CST}$ experiment (from Figure 14). The reduction $\mathcal{B}$ emulates the steps of the $\mathsf{aCh}_{\mathsf{EtS}}$ construction (cf. Figure 13) and uses its sending and receiving oracles from the $\mathsf{CON\text{-}CST}$ game to perform Send and Recv operations. However, when $\mathcal{A}$ queries the first deviating fragment c, $\mathcal{B}$ queries to its $\mathcal{O}_{\mathsf{Recv}}$ oracle the longest genuine prefix $\widetilde{c}$ of c and halts (terminating the simulation). We give the explicit code of algorithm $\mathcal{B}$ in Figure 16.

To see that $\mathcal{B}$ perfectly simulates the oracles of the $\mathsf{aINT\text{-}CST}$ experiment for $\mathcal{A}$ up to the *bad* event occurring, note that $\mathcal{B}$ essentially uses its $\mathcal{O}_{\mathsf{Send}}$ and $\mathcal{O}_{\mathsf{Recv}}$ oracles as a drop-in replacement for the Send and Recv operations performed by the channel $\mathsf{aCh}_{\mathsf{EtS}}$, and executes the (public) encoding and decoding operations by itself.

It remains to argue that if *bad* occurs then $\mathcal{B}$ violates the $\mathsf{CON\text{-}CST}$ property of Ch. To this end, recall that the instructions specified in lines 45–47 identify the first deviating query c and perform an auxiliary invocation of Recv on input the longest genuine prefix $\widetilde{c}$, the latter yielding a (potentially empty) string $\widetilde{v}$. Now, in the $\mathsf{CON\text{-}CST}$ experiment (see Figure 14 for reference) for $\mathcal{B}$, once $\mathcal{B}$ has posed its last query we have that the sequence of sent ciphertexts and the strings of received ciphertext fragments, sent stream message fragments, and received stream message fragments coincide with $\mathbf{C_S}$, $C_R \parallel \widetilde{c}$, $\parallel \mathbf{V_S}$, and $V_R \parallel \widetilde{v}$, respectively, from the experiment E^1. By definition of *bad*, $\mathcal{B}$'s receiving queries cause Recv to output the full stream message string $\parallel \mathbf{V_S}[1,\ldots,j]$ although only a strict prefix of the corresponding ciphertext sequence $\mathbf{C_S}[1,\ldots,j]$ has been submitted for decryption, i.e., $\parallel \mathbf{C_S}[1,\ldots,j] \not\preccurlyeq C_R \parallel \widetilde{c}$ and $\parallel \mathbf{V_S}[1,\ldots,j] \preccurlyeq V_R \parallel \widetilde{v}$. This precisely violates the $\mathsf{CON\text{-}CST}$ property of Ch and hence we have

$$\Pr[bad] \leq \mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda).$$

We finally note that any $\mathcal{A}$ which is successful in experiment E^1 (from Figure 15) can be turned into an efficient adversary $\mathcal{C}$ that breaks the $\mathsf{INT\text{-}CST}$ property of Ch. Similarly to $\mathcal{B}$, algorithm $\mathcal{C}$ simulates

```
$\mathcal{B}^{\mathcal{A},\mathcal{O}_{\mathsf{Send}}(\cdot,\cdot),\mathcal{O}_{\mathsf{Recv}}(\cdot)}(1^\lambda)$:
1  $\mathsf{buf} \leftarrow \varepsilon, \mathsf{fail} \leftarrow 0$
2  $\mathsf{sync} \leftarrow 1$
3  $i \leftarrow 0$
4  $\mathbf{M_S} \leftarrow \mathbf{C_S} \leftarrow ()$
5  $\mathbf{M_R} \leftarrow (), C_R \leftarrow \varepsilon$
6  $\mathcal{A}^{\mathcal{O}^*_{\mathsf{Send}}(\cdot),\mathcal{O}^*_{\mathsf{Recv}}(\cdot)}(1^\lambda)$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Send}}(m)$:
7  $v \leftarrow \mathsf{Encode}(m)$
8  $c \leftarrow \mathcal{O}_{\mathsf{Send}}(v, 1)$
9  $i \leftarrow i + 1$
10 $\mathbf{M_S}[i] \leftarrow m$
11 $\mathbf{C_S}[i] \leftarrow c$
12 return $c$ to $\mathcal{A}$

If $\mathcal{A}$ queries $\mathcal{O}^*_{\mathsf{Recv}}(c)$:
13 if $\mathsf{fail} = 1$ then return $\perp$ to $\mathcal{A}$
14 if $\mathsf{sync} = 1$ and $C_R \,\|\, c \not\preccurlyeq \|\mathbf{C_S}$ and $\|\mathbf{C_S} \not\preccurlyeq C_R \,\|\, c$ then
15     $\widetilde{c} \leftarrow [C_R \,\|\, c, \|\mathbf{C_S}] \,\%\, C_R$
16     $\widetilde{v} \leftarrow \mathcal{O}_{\mathsf{Recv}}(\widetilde{c})$
17     halt
18 $v \leftarrow \mathcal{O}_{\mathsf{Recv}}(c)$
19 $\ell = \max\{|u| : u \preccurlyeq v \wedge u \in \{0,1\}^*\}$
20 $v' \leftarrow s[1..\ell]$
21 $\mathsf{buf} \leftarrow \mathsf{buf} \,\|\, v'$
22 $(\mathbf{m}, \mathsf{buf}) \leftarrow \mathsf{Decode}(\mathsf{buf})$
23 if $v' \neq v$ then
24     $\mathsf{fail} \leftarrow 1, \mathbf{m} \leftarrow \mathbf{m} \,\|\, (\perp)$
25 $C_R \leftarrow C_R \,\|\, c$
26 $\mathbf{M_R} \leftarrow \mathbf{M_R} \,\|\, \mathbf{m}$
27 return $\mathbf{m}$ to $\mathcal{A}$
```

Figure 16: Reduction $\mathcal{B}$ from the aINT-CST of $\mathsf{aCh_{EtS}}$ to the CON-CST of Ch used in the proof of Theorem 7.9. Note that $\mathcal{B}$ does nothing more than emulating the construction (see Figure 13) until the first deviating query. Specifically, it deviates from emulating the construction only with instructions 14–17.

the $\mathsf{aCh_{EtS}}$ operations using the oracles provided by the INT-CST experiment to perform Send and Recv on message fragments. In contrast to $\mathcal{B}$ (which halts when $\mathcal{A}$ poses the first deviating query), $\mathcal{C}$'s simulation goes on until $\mathcal{A}$ stops.

As for $\mathcal{B}$, it is immediate to see that $\mathcal{C}$ perfectly emulates the oracles for $\mathcal{A}$. The more challenging part of the proof is to show that if $\mathcal{A}$ is successful, so is $\mathcal{C}$. More specifically, we show that if $\mathcal{A}$ wins, by triggering lines 40 or 54 in game E^1 (from Figure 15) but not the *bad* event (lines 49 and 11), then $\mathcal{C}$ wins through triggering lines 18 or 35 in the INT-CST game (from Figure 5).

In the rest of the proof we will use the properties of the $\mathsf{aCh_{EtS}}$ construction. First, aSend always invokes Send with flush flag set to $f = 1$: this induces a natural correspondence between $\mathbf{M_S}$, $\mathbf{V_S}$ and $\mathbf{C_S}$. Second, upon processing a receiving query c and the corresponding string v output by Recv in experiment E^1, the sequence $\mathbf{M_R}$ is updated by first appending the message vector $\mathbf{m}$ obtained by decoding the *valid* part v' of fragment v and then, if v contains any error, by also appending the symbol $\perp$ (see lines 26–29 in Figure 15). Moreover, once the first error occurs, $\mathbf{M_R}$ is only further augmented with errors (see line 23), so $\mathcal{A}$ cannot win after this point.

We already saw that there are essentially four possibilities for $\mathcal{A}$ to win the aINT-CST experiment (and so in game E^1). We next show that, assuming that $\mathcal{A}$ is successful, in each of these cases $\mathcal{C}$ violates the INT-CST property of Ch. Let c denote the receiving query (input) that causes $\mathsf{win} \leftarrow 1$ to be set the first time.

We consider first the case in which $\mathsf{sync} \leftarrow 0$ is set with some query prior to c ($\mathcal{A}$ wins by triggering instruction 40).

Case #1. Synchronization has been lost before. Then C_R must be deviating from $\|\mathbf{C_S}$ (see footnote 17). Note that for $\mathcal{A}$ to win, aRecv on input the (fully deviating) fragment c must output some message sequence $\mathbf{m} \neq (\perp)$. That is, if buf denotes the buffer kept by aRecv prior to processing c, we have that Recv on input c returns a string v such that $\mathsf{Decode}(\mathsf{buf} \,\|\, v') = (\mathbf{m}, \mathsf{buf}')$, where v' is the longest error-free prefix of v. In other words, processing c augments the buffer to contain (at least) a full codeword v^* that was not completed before receiving c (otherwise $\mathsf{win} \leftarrow 1$ would have been set with some previous query).

In more detail, let c'_1 denote the first deviating query, let $c'_2, \ldots, c'_\ell$ be the subsequent decryption queries prior to c, and let $v'_1, \dots, v'_\ell$ be the output of Recv on input these queries. Since $\mathsf{win} \leftarrow 1$ is only set when $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}$ on c, the sequence of messages received after processing each of $c'_1, \dots, c'_\ell$ must be a prefix of $\mathbf{M_S}$, i.e., $\mathbf{M_R} = \mathbf{M_S}[1, \dots, k]$ for some $1 \leq k < i$, and moreover $v'_1, \dots, v'_\ell \in \{0,1\}^*$ do not contain an error symbol. Prior to receiving c we must have that $\| \mathbf{V_S}[1, \dots, k] \preccurlyeq V_R$ and the remainder $V_R \,\%\, \| \mathbf{V_S}[1, \dots, k] = v''_1 \parallel v'_2 \parallel \cdots \parallel v'_\ell$, where v''_1 is a suffix of v'_1 (some genuine prefix of v'_1 may complete the codeword $\mathbf{V_S}[k]$), does not contain any full codeword (again, otherwise $\mathsf{win} \leftarrow 1$ would be set earlier). Only when c is processed and its output v' appended to the current buffer can we isolate a full codeword v^*, i.e., $v^* \preccurlyeq \mathsf{buf} = v''_1 \parallel \cdots \parallel v'_\ell \parallel v'$ such that $\mathsf{Decode}(v^*) = (\mathbf{M_R}[k+1], \varepsilon)$. In particular, $v^* \notin \mathcal{E}^*$. Thus, some among the deviating queries $c'_1, \dots, c'_\ell, c$ that $\mathcal{C}$ forwards to its own receiving oracle will make $\mathcal{C}$ win in the $\mathsf{INT\text{-}CST}$ experiment by triggering line 18 or line 35 in Figure 5.

The next three cases are those induced by the clauses defining the predicate of line 25 in the $\mathsf{aINT\text{-}CST}$ experiment (Figure 11). Here $\mathsf{win} \leftarrow 1$ and $\mathsf{sync} \leftarrow 0$ are set within the same $\mathcal{O}_{\mathsf{Recv}}$ call in E^1 (in lines 51 and 54). Note that for the $\mathsf{aINT\text{-}CST}$ experiment to enter the loop of lines 25–26 the current query c must cause $C_R \not\preccurlyeq \| \mathbf{C_S}$. The three cases to consider are the following.

Case #2. C_R goes *ahead* of $\| \mathbf{C_S}$ (the loop terminates because $j > i$). Then after c is processed we have $j = i + 1$, $\| \mathbf{C_S} \prec C_R$ and $\mathbf{M_S} \preccurlyeq \mathbf{M_R}$. Assuming that $\mathcal{A}$ wins the game we must additionally have $\mathbf{M_S}[1, \dots, i] \prec \mathbf{M_R}$ and in particular $\mathbf{M_R}[j] \neq \bot$. It follows from the conditions above that $V_R = \| \mathbf{V_S}[1, \dots, i] \parallel v^* \parallel s$ such that $\mathsf{Decode}(v^*) = (\mathbf{M_R}[j], \varepsilon)$.[18] That is, Recv outputs on input c a string $v' \parallel v^* \parallel s \notin \mathcal{E}^*$, where v' might complete previous fragments. Hence, this triggers instruction 35 (in Figure 5), making $\mathcal{C}$ violate the $\mathsf{INT\text{-}CST}$ property of Ch.

Case #3. The stream C_R *deviates* from $\| \mathbf{C_S}$ after the first $j - 1$ sent ciphertexts and messages are received entirely (the loop terminates because $j \leq i$ but $\| \mathbf{C_S}[1, \dots, j] \not\preccurlyeq C_R$). In this case we have $\| \mathbf{C_S}[1, \dots, j-1] \prec C_R$ and $\mathbf{M_S}[1, \dots, j-1] \preccurlyeq \mathbf{M_R}$, but $\| \mathbf{C_S}[1, \dots, j] \not\preccurlyeq C_R$. Again assuming that $\mathsf{win} \leftarrow 1$ is set with the current query we have $\mathbf{M_S}[1, \dots, j-1] \prec \mathbf{M_R}$ and $\mathbf{M_R}[j] \neq \bot$. Then $V_R = \| \mathbf{V_S}[1, \dots, j-1] \parallel v^* \parallel s$ with $v^* \in \{0,1\}^*$, $s \in (\{0,1\} \cup \mathcal{E})^*$ and $\mathsf{Decode}(v^*) = (\mathbf{M_R}[j], \varepsilon)$. This means that on processing the current (deviating) query the stream-based algorithm Recv returns (beyond a potential genuine prefix) a non-error output $v = v^* \parallel s$, i.e., $v \notin \mathcal{E}^*$. Now, if $\mathbf{M_R}[j] \neq \mathbf{M_S}[j]$ we have $v^* \neq \mathbf{V_S}[j]$ and, because of the prefix-freeness of the set of codewords, that $v^* \not\preccurlyeq \mathbf{V_S}[j]$, which is an evident violation of the $\mathsf{INT\text{-}CST}$ property. The case $\mathbf{M_R}[j] = \mathbf{M_S}[j]$ requires more care. Indeed, since $v^* = \mathbf{V_S}[j]$, we cannot directly argue that having $v^* \in \{0,1\}^*$ violates the $\mathsf{INT\text{-}CST}$ property of Ch, as v^* may be also output by Recv when processing the *genuine* prefix $\tilde{c}$ of c. However, the latter would imply $v^* = \mathbf{V_S}[j] \preccurlyeq \tilde{v}$, hence triggering the *bad* event (instructions 49 and 11 in Figure 15), meaning that $\mathcal{A}$ loses the game, against our assumption. That is, Recv only outputs the full v^* after processing the deviating part of c, hence violating the $\mathsf{INT\text{-}CST}$ security of Ch.

Case #4. The sequence $\mathbf{M_R}$ *deviates* from $\mathbf{M_S}$ after the first $j-1$ sent messages are received completely (the loop terminates because $j \leq i$ and $\| \mathbf{C_S}[1, \dots, j] \preccurlyeq C_R$ but $\mathbf{M_S}[1, \dots, j] \not\preccurlyeq \mathbf{M_R}[1, \dots, j]$). As $\mathsf{win} \leftarrow 1$ is set with the current query we have $\mathbf{M_S}[1, \dots, j-1] \preccurlyeq \mathbf{M_R}$, $\mathbf{M_S}[1, \dots, j] \not\preccurlyeq \mathbf{M_R}$ and $\mathbf{M_R}[j] \neq \bot$. Then $V_R = \| \mathbf{V_S}[1, \dots, j-1] \parallel v^* \parallel s$ for some $v^* \in \{0,1\}^*$ and $s \in (\{0,1\} \cup \mathcal{E})^*$ such that $\mathsf{Decode}(v^*) = (\mathbf{M_R}[j], \varepsilon)$, where $v^* \neq \mathbf{V_S}[j]$ and, for the prefix-freenes, $v^* \not\preccurlyeq \mathbf{V_S}[j]$, which again violates the $\mathsf{INT\text{-}CST}$ of Ch.

[18]In fact, the implication $\mathbf{M_S} \preccurlyeq \mathbf{M_R} \implies \| \mathbf{V_S} \preccurlyeq V_R$ holds under the assumption that only codewords are decoded to valid messages (i.e., there exists no string $x \in \{0,1\}^* \setminus V$ such that $\mathsf{Decode}(x) = (\mathbf{m}, s)$ with $\mathbf{m} \in \mathcal{M}^*$). For the argument to go through, however, we do not need such an assumption, as having $\| \mathbf{V_S} \not\preccurlyeq V_R$ here would immediately lead to a violation of the $\mathsf{INT\text{-}CST}$ property of Ch. A similar argument also applies to cases #3 and #4.

The analysis above proves that

$$\mathsf{Adv}^{\mathsf{E}^1}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda).$$

Overall, we hence obtain the final bound

$$\mathsf{Adv}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{C}}(\lambda). \qquad \square$$

We saw in Theorem 7.9 that the encode-then-stream paradigm allows us to leverage integrity and conciseness of ciphertext streams for stream-based channels (INT-CST and CON-CST) to integrity of ciphertexts for atomic-message channels (aINT-CST). In the following theorems we prove a similar result for confidentiality. First, we show in Theorem 7.10 that the encode-then-stream paradigm lifts confidentiality against chosen plaintext-fragment attacks (IND-CPFA) to confidentiality against (atomic) chosen-plaintext attacks (aIND-CPA). Then, we prove in Theorem 7.11 that $\mathsf{aCh}_{\mathsf{EtS}}$ maintains (in the atomic setting) the error predictability of the underlying stream-based channel. Finally, as a corollary of the above results together with Theorem 6.6, we conclude that CON-CST, INT-CST, IND-CPFA, and ERR-PRE of Ch imply aIND-CCFA security of $\mathsf{aCh}_{\mathsf{EtS}}$.

Theorem 7.10 (aIND-CPA security of $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and let* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 7.5). If* Ch *provides and indistinguishability under chosen plaintext-fragment attacks (*IND-CPFA*) then* $\mathsf{aCh}_{\mathsf{EtS}}$ *provides indistinguishability under chosen-plaintext attacks in the atomic-message setting (*aIND-CPA*). Formally, for every efficient* aIND-CPA *adversary* $\mathcal{A}$ *there exist an efficient* IND-CPFA *adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{aIND\text{-}CPA}}_{\mathsf{aCh}_{\mathsf{EtS}},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPFA}}_{\mathsf{Ch},\mathcal{B}}.$$

Proof. Given a left-or-right oracle from the IND-CPFA experiment against Ch it is immediate to simulate the aIND-CPA experiment. Whenever $\mathcal{A}$ queries (m_0, m_1), algorithm $\mathcal{B}$ computes $v_0 = \mathsf{Encode}(m_0)$, $v_1 = \mathsf{Encode}(m_1)$, queries$(v_0, v_1, 1)$ to its own oracle $\mathcal{O}_{\mathsf{LoR}}$, and gives the result c to $\mathcal{A}$. Observe that $|v_0| = |v_1|$ due to the length-regularity of ES, so the queries made by $\mathcal{B}$ to its oracle are valid. As soon as $\mathcal{A}$ stops and returns a bit b', so does $\mathcal{B}$. Hence, analogously to the reductions described in the proof of Theorem 7.9, $\mathcal{B}$ perfectly emulates the oracle for $\mathcal{A}$, executing the streaming Send operation via its oracle. Thus, $\mathcal{B}$ has the same advantage as $\mathcal{A}$. $\square$

Theorem 7.11 (aERR-PRE security of $\mathsf{aCh}_{\mathsf{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and let* $\mathsf{aCh}_{\mathsf{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 7.5). If* Ch *provides error predictability (*ERR-PRE*) with respect to some efficient predictor* Pred *then there exists an efficient predictor* Pred′ *(described in the proof) such that* $\mathsf{aCh}_{\mathsf{EtS}}$ *is error predictable (*aERR-PRE*) with respect to* Pred′*. Formally, for every efficient* aERR-PRE *adversary* $\mathcal{A}$ *there exists an efficient* ERR-PRE *adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\mathsf{aERR\text{-}PRE}}_{\mathsf{aCh}_{\mathsf{EtS}},\mathsf{Pred}',\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ERR\text{-}PRE}}_{\mathsf{Ch},\mathsf{Pred},\mathcal{B}}.$$

Proof. Let Pred′ be an algorithm that on input $\mathbf{C_S}$, C_R, and c (as described in the experiment from Figure 12) invokes Pred as a subroutine on input $\| \mathbf{C_S}$, C_R, and c. Then, depending on the output e of Pred, Pred′ returns an empty vector () if $e = \varepsilon$ is the empty string, otherwise it returns $(\perp)$.

To see that Pred′ is a valid error predictor for $\mathsf{aCh}_{\mathsf{EtS}}$, notice first that by construction algorithm aRecv always outputs an element $\mathbf{m}$ in $\mathcal{M}^* \cup (\mathcal{M}^* \times \perp)$, i.e., a vector of messages possibly followed by the error symbol $\perp$. Thus, for the projection $\langle \mathbf{m} \rangle_{\mathcal{E}}$ there are only two possibilities: it is either the empty vector () or the singleton $(\perp)$. Now let $\mathcal{A}$ be a successful adversary in the $\mathsf{aERR\text{-}PRE}$ experiment (from Figure 12) against the error predictor Pred'. Then we can build an algorithm $\mathcal{B}$, to be run in the $\mathsf{ERR\text{-}PRE}$ experiment (from Figure 6) against Pred, similarly to the reductions described in Theorems 7.9 and 7.10, i.e., by letting $\mathcal{B}$ emulate the oracles for $\mathcal{A}$ by performing the public encoding and decoding operations of $\mathsf{aCh_{EtS}}$ and using its sending and receiving oracles for the streaming operations.

There are only two possibilities for $\mathcal{A}$ to be successful (in line 13 of Figure 12): either (i) Pred' returns () while aRecv outputs an error, thus $\langle \mathbf{m} \rangle_{\mathcal{E}} = (\perp)$, or (ii) Pred' returns $(\perp)$ but aRecv only produces valid messages, thus $\langle \mathbf{m} \rangle_{\mathcal{E}} = ()$. Observe that by construction Pred' returns an empty vector if and only if Pred returns an empty string. Similarly, aRecv outputs $\perp$ if and only if Recv also returns some error symbols. We hence deduce that, in the first case, Recv on input c returns error symbols but Pred erroneously predicts that no error occurred, causing the execution of line 11. The second case is analogous: Recv on input c only produces valid message bits, while Pred falsely predicts errors. In either case, $\mathcal{B}$ wins by causing the execution of line 11 in Figure 6. □

Corollary 7.12 ($\mathsf{aIND\text{-}CCFA}$ security of $\mathsf{aCh_{EtS}}$)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and let* $\mathsf{aCh_{EtS}} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be the atomic-message channel obtained from* Ch *via the encode-then-stream construction (see Construction 7.5). If* Ch *provides integrity and conciseness of ciphertext streams (*$\mathsf{INT\text{-}CST}$ *and* $\mathsf{CON\text{-}CST}$*), error predictability (*$\mathsf{ERR\text{-}PRE}$*), and indistinguishability under a chosen plaintext-fragment attack (*$\mathsf{IND\text{-}CPFA}$*), then* $\mathsf{aCh_{EtS}}$ *provides indistinguishability under a chosen-ciphertext attack in the atomic-message setting (*$\mathsf{aIND\text{-}CCFA}$*).*

7.3.3 Secure Instantiation from AEAD-based Streaming Channel Construction

The results of the previous section establish sufficient security requirements on the underlying stream-based channel to obtain a secure atomic-message channel via the encode-then-stream paradigm. One of these requirements is conciseness of ciphertext streams ($\mathsf{CON\text{-}CST}$), which essentially says that ciphertext fragments produced by the streaming sending algorithm contain no redundant bits that can be chopped or modified without altering the underlying message fragment. It is not difficult to achieve this property. Indeed, as we show in the next theorem, our AEAD-based streaming channel $\mathsf{Ch_{AEAD}}$ from Construction 5.1 provides conciseness of ciphertext streams. This result holds unconditionally in an information-theoretic sense; even an unbounded adversary cannot violate $\mathsf{CON\text{-}CST}$ of $\mathsf{Ch_{AEAD}}$.

Theorem 7.13 ($\mathsf{CON\text{-}CST}$ of $\mathsf{Ch_{AEAD}}$)**.** *The stream-based channel* $\mathsf{Ch_{AEAD}}$ *from Construction 5.1 unconditionally provides conciseness of ciphertext streams (*$\mathsf{CON\text{-}CST}$*). Formally, for any (even unbounded)* $\mathsf{CON\text{-}CST}$ *adversary* $\mathcal{A}$ *against* $\mathsf{Ch_{AEAD}}$ *it holds that*

$$\mathsf{Adv}^{\mathsf{CON\text{-}CST}}_{\mathsf{Ch_{AEAD}},\mathcal{A}} = 0.$$

Proof. Observe that a genuine ciphertext fragment c produced by Send on input some message fragment $m \in \{0,1\}^*$ is, by construction, the concatenation of a number of blocks $B_1, B_2, \ldots$, each containing an AEAD ciphertext c'_i preceded by (the bit representation of) its length len_i. Importantly, Recv identifies these blocks in the received ciphertext stream and only upon receiving a *full* block $B_i = len_i \parallel c'_i$ does it invoke the AEAD decryption algorithm on ciphertext c'_i. Thus, it is impossible to violate conciseness of ciphertexts. Indeed, recall from the $\mathsf{CON\text{-}CST}$ game (see Figure 14) that in order to win the adversary can only submit to $\mathcal{O}_{\mathsf{Recv}}$ a strict prefix of the genuine ciphertext stream. However, in the case of $\mathsf{Ch_{AEAD}}$ not even an unbounded adversary can make Recv output one of the sent message fragments *entirely* by submitting a truncation of the corresponding ciphertext fragment: chopping the ciphertext prevents the AEAD decryption from being invoked on some block B_i and, consequently, causes a truncation of the genuine message stream. □

Remark 7.14. The TLS record protocol in both versions 1.2 and 1.3 [DR08, Res17] also provides conciseness of its ciphertext stream when using an AEAD scheme and omitting the option to send zero-length message fragments. As in our construction, TLS records are data structures clearly marked-out via a length header, forming blocks within the ciphertext fragments output by the sender. Hence, for reasons similar to those in Theorem 7.13, even an unbounded adversary cannot force entire message fragments to be output when receiving truncated ciphertext fragments.

Theorem 7.13 together with the already established $\mathsf{INT\text{-}CST}$ and $\mathsf{IND\text{-}CCFA}$ security of $\mathsf{Ch}_{\mathsf{AEAD}}$ under the mild assumption that the authenticated encryption scheme with associated data AEAD provides AUTH and $\mathsf{IND\text{-}CPA}$ security suggests that $\mathsf{Ch}_{\mathsf{AEAD}}$ is a 'good' stream-based channel to start with for building secure atomic-message channels using the encode-then-stream transform. Moreover, the atomic-message channel construction $\mathsf{aCh}_{\mathsf{EtS}}$ applied to $\mathsf{Ch}_{\mathsf{AEAD}}$ yields a secure atomic-message channel from AEAD.

Corollary 7.15 (Atomic-message channels from AEAD)**.** *Let* AEAD *be an authenticated encryption scheme with associated data, let* $\mathsf{Ch}_{\mathsf{AEAD}}$ *be the streaming channel obtained from* AEAD *via Construction 5.1, and let* $\mathsf{aCh}_{\mathsf{EtS}}$ *be the atomic-message channel construction from Construction 7.5 applied to* $\mathsf{Ch}_{\mathsf{AEAD}}$*. If* AEAD *provides authenticity (*AUTH*) and indistinguishability under chosen-plaintext attacks (*$\mathsf{IND\text{-}CPA}$*) then* $\mathsf{aCh}_{\mathsf{EtS}}$ *provides atomic integrity of ciphertext streams (*$\mathsf{aINT\text{-}CST}$*) and atomic indistinguishability under chosen ciphertext-fragment attacks (*$\mathsf{aIND\text{-}CCFA}$*).*

8 Conclusion and Open Problems

In this work we studied the security of channels designed to (securely) convey a stream of data from one party to another, narrowing the gap between real-world transport layer security protocols (like TLS or SSH) and our theoretical understanding of them. For this purpose, we formalized the syntax of such stream-based channels, explored strong security notions, and demonstrated their feasibility by providing a natural and secure construction which closely mimics the operation of the TLS record protocol. We furthermore analyzed how applications can securely transport atomic messages over a stream-based channel by providing a provably secure 'encode-then-stream' paradigm.

Our approach sheds a formal light on recent attacks, in particular concerning the use of HTTP over TLS, confirming a disjunction between applications' expectations on the one hand and the guarantees that secure streaming channels provide on the other. This highlights that there is a need for detailed specifications of APIs and security guarantees for such protocols.

Our work also raises new research questions. Naturally, exploring the exact relation between stream-based and atomic-message channels is an avenue that should be pursued, with the development of detailed relations between security notions in our work and those by Boldyreva et al. [BDPS12] as a specific task. Considering established techniques, the open question remains whether the well-accepted concept of length-hiding encryption can be incorporated in the stream-based setting despite being intrinsically connected to atomic messages. It also seems worthwhile to extend our stream-based model to encompass channel protocol designs (such as TLS and QUIC) that allow multiplexing of several data streams within a single channel.

Some technical questions arise from our analysis of the encode-then-stream paradigm to build atomic-message channels from stream-based channels generically. Our security proofs for integrity and confidentiality against active adversaries both rely on the conciseness of ciphertext streams ($\mathsf{CON\text{-}CST}$) of the underlying stream-based channel. A natural question is whether $\mathsf{CON\text{-}CST}$ is necessary for proving security of the encode-then-stream construction and, more generally, whether it is necessary for the security of *any* black-box construction of an atomic-message channel running on top of a stream-based channel in a network-layer sense.

As we explain in Appendix B, there exists a class of stream-based channels that do not provide integrity protection but are intuitively confidential, yet are declared insecure in our model. This raises the challenging question whether a weaker, yet reasonable, notion of confidentiality can be formalized for stream-based channels which is met by the above-mentioned class of channels.

Finally, our work raises an important conceptual question for the practical design of secure channels: Which type of interface should a channel provide in practice? A streaming one, following the historic tradition of TCP and the de-facto standard socket interface for network protocols, or an atomic-message interface, ensuring strong cryptographic protection of message boundaries? In the very end, the most versatile approach might be to provide both interfaces, one for streaming data and one for distinct-message data, and allow the application to select the one that best suits its specific needs.

Acknowledgments

The authors thank Jean Paul Degabriele for communicating the class of trivial attacks that both [BDPS12] and the proceedings version of this work [FGMP15] failed to capture, and Bertram Poettering for informing us about an instance of an intuitively confidential scheme that is declared insecure in our model; we thank both for helpful discussions. Further, the authors thank the anonymous reviewers for their valuable comments.

Marc Fischlin was supported by the Heisenberg grant Fi 940/3-2 of the German Research Foundation (DFG). Kenneth Paterson was supported by EPSRC Leadership Fellowship EP/H005455/1, by EPSRC grants EP/M013472/1, EP/K035584/1 and EP/P009301/1, and by a research programme funded by Huawei Technologies and delivered through the Institute for Cyber Security Innovation at Royal Holloway. This work has been co-funded by the DFG as part of projects P2 and S4 within the CRC 1119 CROSSING and as part of the priority program 1736 Algorithms for Big Data, as well as by the EU COST Action IC 1306.

Most of the work on this paper was done while Giorgia Azzurra Marson was at TU Darmstadt and at Ruhr University Bochum.

References

[ADHP16] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 1480–1491, Vienna, Austria, October 24–28, 2016. ACM Press. (Cited on pages 4, 5, 16, 17, 31, 32, and 34.)

[APW09] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press. (Cited on page 4.)

[Bar07] Gregory V. Bard. Blockwise-adaptive chosen-plaintext attack and online modes of encryption. In Steven D. Galbraith, editor, *11th IMA International Conference on Cryptography and Coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 129–151, Cirencester, UK, December 18–20, 2007. Springer, Heidelberg, Germany. (Cited on page 7.)

[BBKN01] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. Online ciphers and the hash-CBC construction. In Joe Kilian, editor, *Advances in Cryptology –*

CRYPTO 2001, volume 2139 of *Lecture Notes in Computer Science*, pages 292–309, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. (Cited on page 7.)

[BDF⁺14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press. (Cited on pages 4, 6, 19, 31, 37, and 42.)

[BDPS12] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. (Cited on pages 4, 5, 7, 11, 12, 13, 16, 17, 19, 31, 32, 33, 37, 50, 51, and 57.)

[BDPS14] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. On symmetric encryption with distinguishable decryption failures. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 367–390, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany. (Cited on pages 6, 19, 20, 30, and 31.)

[Ber] Dan Bernstein. Cryptographic competitions: CAESAR. `http://competitions.cr.yp.to/caesar.html`. (Cited on page 7.)

[BFK⁺13] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *2013 IEEE Symposium on Security and Privacy*, pages 445–459, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press. (Cited on page 6.)

[BKN02] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 1–11, Washington D.C., USA, November 18–22, 2002. ACM Press. (Cited on page 3.)

[BKN04] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. *ACM Trans. Inf. Syst. Secur.*, 7(2):206–241, 2004. (Cited on pages 3, 4, 5, 11, 12, 17, 19, 24, 31, and 57.)

[BMM⁺15] Christian Badertscher, Christian Matt, Ueli Maurer, Phillip Rogaway, and Björn Tackmann. Augmented secure channels and the goal of the TLS 1.3 record layer. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015: 9th International Conference on Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 85–104, Kanazawa, Japan, November 24–26, 2015. Springer, Heidelberg, Germany. (Cited on page 3.)

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany. (Cited on pages 6 and 19.)

[BPS15a] Guy Barwell, Dan Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. Cryptology ePrint Archive, Report 2015/895, 2015. `http://eprint.iacr.org/2015/895`. (Cited on page 24.)

[BPS15b] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. In Jens Groth, editor, *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *Lecture Notes in Computer Science*, pages 94–111, Oxford, UK, December 15–17, 2015. Springer, Heidelberg, Germany. (Cited on pages 24 and 30.)

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany. (Cited on page 21.)

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 373–386, Berlin, Germany, November 4–8, 2013. ACM Press. (Cited on page 31.)

[BT04] Alexandra Boldyreva and Nut Taesombut. Online encryption schemes: New security notions and constructions. In Tatsuaki Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 1–14, San Francisco, CA, USA, February 23–27, 2004. Springer, Heidelberg, Germany. (Cited on page 7.)

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`. (Cited on page 3.)

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany. (Cited on page 3.)

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. (Cited on pages 16 and 40.)

[Deg16] Jean Paul Degabriele. Personal communication, May 2016. (Cited on pages 13 and 16.)

[DLFK$^+$17] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy*, pages 463–482, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press. (Cited on page 7.)

[DP10] Jean Paul Degabriele and Kenneth G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 493–504, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. (Cited on page 4.)

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176. (Cited on pages 3, 4, 8, 25, 30, and 50.)

[FFL12] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany. (Cited on page 7.)

[FGMP15] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. (Cited on pages 7, 13, 16, 18, and 51.)

[FJMV04] Pierre-Alain Fouque, Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Authenticated on-line encryption. In Mitsuru Matsui and Robert J. Zuccherato, editors, *SAC 2003: 10th Annual International Workshop on Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159, Ottawa, Ontario, Canada, August 14–15, 2004. Springer, Heidelberg, Germany. (Cited on page 7.)

[FJP04] Pierre-Alain Fouque, Antoine Joux, and Guillaume Poupard. Blockwise adversarial model for on-line ciphers and symmetric encryption schemes. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004: 11th Annual International Workshop on Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 212–226, Waterloo, Ontario, Canada, August 9–10, 2004. Springer, Heidelberg, Germany. (Cited on page 7.)

[FR14] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard), June 2014. (Cited on pages 37 and 39.)

[HRRV15] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 493–517, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. (Cited on page 7.)

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. (Cited on pages 3, 15, and 31.)

[JMV02] Antoine Joux, Gwenaëlle Martinet, and Frédéric Valette. Blockwise-adaptive attackers: Revisiting the (in)security of some provably secure encryption models: CBC, GEM, IACBC. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 17–30, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. (Cited on page 7.)

[KPB03] Tadayoshi Kohno, Adriana Palacio, and John Black. Building secure cryptographic transforms, or how to encrypt and MAC. Cryptology ePrint Archive, Report 2003/177, 2003. `http://eprint.iacr.org/2003/177`. (Cited on pages 3 and 31.)

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. (Cited on pages 3 and 15.)

[KS05] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040. (Cited on page 3.)

[MT10] Ueli Maurer and Björn Tackmann. On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 505–515, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. (Cited on page 3.)

[Nam02] Chanathip Namprempre. Secure channels based on authenticated encryption schemes: A simple characterization. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 515–532, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany. (Cited on page 3.)

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. (Cited on pages 19 and 41.)

[oEEE] IEEE (Institute of Electrical and Inc.) Electronics Engineers. IEEE Standard 801.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. (Cited on page 3.)

[Ope] The OpenSSL Project. `https://www.openssl.org`. (Cited on page 42.)

[OS05] Rafail Ostrovsky and William E. Skeith III. Private searching on streaming data. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. (Cited on page 7.)

[Poe16] Bertram Poettering. Personal communication, July 2016. (Cited on pages 16 and 60.)

[Pos80] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980. (Cited on pages 3 and 8.)

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528. (Cited on pages 3 and 8.)

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. (Cited on pages 3, 6, 15, and 31.)

[PW10] Kenneth G. Paterson and Gaven J. Watson. Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 345–361, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. (Cited on page 4.)

[PY14] Periklis A. Papakonstantinou and Guang Yang. Cryptography with streaming algorithms. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 55–70, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on page 7.)

[QUI] QUIC, a multiplexed stream transport over UDP. `https://www.chromium.org/quic`. Retrieved on 2017-04-21. (Cited on pages 3 and 8.)

[Res15] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-11. `https://tools.ietf.org/html/draft-ietf-tls-tls13-11`, December 2015. (Cited on page 25.)

[Res17] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-22. `https://tools.ietf.org/html/draft-ietf-tls-tls13-22`, November 2017. (Cited on pages 30 and 50.)

[rGPP] 3GPP (3rd Generation Partnership Project). GSM, UMTS, and LTE standards. `http://www.3gpp.org`. (Cited on page 3.)

[RM12] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012. (Cited on page 3.)

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 02: 9th Conference on Computer and Communications Security*, pages 98–107, Washington D.C., USA, November 18–22, 2002. ACM Press. (Cited on pages 3, 5, 24, 25, and 30.)

[Sho99] Victor Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. `http://eprint.iacr.org/1999/012`. (Cited on page 3.)

[Sho06] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs, 2006. Manuscript. (Cited on page 21.)

[SP13] Ben Smyth and Alfredo Pironti. Truncating TLS connections to violate beliefs in web applications. In *WOOT'13: 7th USENIX Workshop on Offensive Technologies*. USENIX Association, 2013. (first appeared at Black Hat USA 2013). (Cited on pages 4, 6, 19, and 31.)

[TSS09] Patrick P. Tsang, Rouslan V. Solomakhin, and Sean W. Smith. Authenticated streamwise online encryption. Technical Report TR2009-640, Department of Computer Science, Dartmouth College, 2009. (Cited on page 7.)

[YL06a] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006. (Cited on page 3.)

[YL06b] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), January 2006. Updated by RFC 6668. (Cited on page 8.)

A Alternative Confidentiality Definition

While the confidentiality notions for stateful encryption schemes with atomic algorithms defined by Bellare et al. [BKN04, Section 6.2], introducing the concept of the decryption oracle being in or out of sync, are well-established today, defining confidentiality in case of fragmented inputs is less obvious. Considering fragmented delivery of atomic ciphertexts, Boldyreva et al. [BDPS12, Section 3.2] decided to stay close to the original definitions by Bellare et al. and conservatively defined synchronization to be lost at ciphertext boundaries (thereby potentially suppressing less output, resulting in a stronger security definition). This approach however is too strong if fragmentation is also to be considered on the sender's side of a channel, as revealing the whole message input corresponding to a ciphertext output by the Send algorithm renders channels that (like, e.g., TLS) do some internal refragmentation generically insecure (see the discussion in Section 4.1).

During the process of formalizing a notion of confidentiality for stream-based channels which is reasonably strong but permissive enough to cover 'naturally secure' constructions, a second variant of confidentiality emerged beyond the one we provide in Section 4.1. Although we consider Definition 4.1 to be more attractive, to make our choice transparent we will present the alternative definition here. We then clarify why Definition 4.1 is superior by highlighting that the alternative notion is only applicable to a restricted class of channels and by formally proving equivalence of the two notions for all channels of that class. A similar reasoning also applies for integrity notions.

Confidentiality definition with split Recv calls. The only difference between the two variants in question is the behavior of the oracle $\mathcal{O}_{\mathsf{Recv}}$ upon processing a ciphertext fragment c that causes C_R to deviate from or go ahead of C_S, but also contains a genuine prefix $\widetilde{c}$ to be stripped off. In the second variant, that we name 'confidentiality with split Recv calls' for differentiation, the receiving oracle $\mathcal{O}'_{\mathsf{Recv}}$ is defined as shown in Figure 17. Concretely, all changes occur within lines 37–40 of the oracle $\mathcal{O}'_{\mathsf{Recv}}$. Essentially, while $\widetilde{c}$ remains the longest genuine prefix of c, here the $\mathcal{O}'_{\mathsf{Recv}}$ oracle first Recv on input $\widetilde{c}$ and suppresses its output, then it invokes Recv a second time *on the now updated state* on input the remaining part of c, namely $c \,\%\, \widetilde{c}$, and this time gives the resulting message fragment m' to the adversary.

Since the modified $\mathcal{O}'_{\mathsf{Recv}}$ oracle only affects the CCFA version of our confidentiality notion, we ignore the IND-CPFA case here.

Definition A.1 (IND-CCFA$'$ Security)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel and experiment* $\mathsf{Expt}^{\mathsf{IND\text{-}CCFA}',b}_{\mathsf{Ch},\mathcal{A}}(1^\lambda)$ *for an adversary* $\mathcal{A}$ *and a bit* b *be defined as in Figure 3 but with the receiving oracle* $\mathcal{O}'_{\mathsf{Recv}}$ *from Figure 17. We say* Ch *provides* indistinguishability under **split-call** chosen ciphertext-fragment attacks *(*IND-CCFA$'$*) if for all PPT adversaries* $\mathcal{A}$ *the following advantage function is negligible in the security parameter:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}',b}_{\mathsf{Ch},\mathcal{A}}(\lambda) := \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}CCFA}',1}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}CCFA}',0}_{\mathsf{Ch},\mathcal{A}}(1^\lambda) = 1\right].$$

Fragmentation-independent behavior. The most important difference between the confidentiality experiments in Figure 3 and the one using the modified oracle $\mathcal{O}'_{\mathsf{Recv}}$ from Figure 17 with split calls to Recv is that the latter implicitly assumes that splitting up the ciphertext fragment c into its genuine part $\widetilde{c}$ and the rest $c' = c \,\%\, \widetilde{c}$, and hence processing sequentially the two parts, does not affect the behavior (i.e., output) of the Recv algorithm. We call this property *fragmentation-independent behavior* of a stream-based channel, and formalize it next.

Definition A.2 (Fragmentation-independent behavior)**.** *We say that a channel* Ch *exhibits* fragmentation-independent behavior *if for all* $\mathsf{st}_R \in \mathcal{S}_R$*, all choices of the randomness for algorithm* Recv *and ciphertext*

```
If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Recv}}(c)$:
if $\mathsf{sync} = 0$ then  // already out-of-sync
    $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
    return $m$ to $\mathcal{A}$
else if $C_R \parallel c \preccurlyeq C_S$ then  // still in-sync
    $(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
    $C_R \leftarrow C_R \parallel c$
    return $\varepsilon$ to $\mathcal{A}$
else
    if $C_R \prec [C_R \parallel c, C_S]$ then
        // $c$ deviates or exceeds, contains genuine part
        $\widetilde{c} \leftarrow [C_R \parallel c, C_S] \,\%\, C_R$
        $c' \leftarrow c \,\%\, \widetilde{c}$
        $(\mathsf{st}_R, \widetilde{m}) \leftarrow \mathsf{Recv}(\mathsf{st}_R, \widetilde{c})$
        $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c')$
    else  // $c$ deviates or exceeds, contains no genuine part
        $(\mathsf{st}_R, m') \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
    if $C_S \not\preccurlyeq C_R \parallel c$ or $m' \neq \varepsilon$ then
        // deviation, or exceeding portion produces output
        $\mathsf{sync} \leftarrow 0$
    $C_R \leftarrow C_R \parallel c$
    return $m'$ to $\mathcal{A}$
```

Figure 17: Alternative definition $\mathcal{O}'_{\mathsf{Recv}}$ of the receiving oracle from the security experiment for *confidentiality* of stream-based channels (see Figure 3) using split calls to Recv.

fragments $\mathbf{c}, \mathbf{c}' \in (\{0,1\}^*)^*$*, it holds* $\parallel \mathbf{c} \preccurlyeq \parallel \mathbf{c}' \implies \parallel \mathbf{m} \preccurlyeq \parallel \mathbf{m}'$*, where* $\mathbf{m}$ *is output by* $\mathsf{Recv}(\mathsf{st}_R, \mathbf{c})$ *and* $\mathbf{m}'$ *is output by* $\mathsf{Recv}(\mathsf{st}_R, \mathbf{c}')$*.*

Remark A.3. By symmetry, Definition A.2 also guarantees that $\parallel \mathbf{c} = \parallel \mathbf{c}'$ implies $\parallel \mathbf{m} = \parallel \mathbf{m}'$.

Intuitively, the Recv algorithm of a channel with fragmentation-independent behavior always generates the same output sequence for some fixed input ciphertext stream, independently of how the ciphertext stream is fragmented. Note that this property is *not* guaranteed by the correctness of the stream-based channel, which only concerns the processing by Recv of streams generated by Send. While fragmentation-independent behavior is a characteristic that many real-world protocols exhibit (e.g., TLS), a scheme could potentially process received fragments in a way which is not fragmentation-independent. Consider, e.g., a variant of TLS which, when given some ciphertext stream fragment c, first decodes all contained records and, if any of them contains an error, it outputs the empty string.[19]

More importantly, without fragmentation-independent behavior the $\mathsf{IND\text{-}CCFA}'$ split-call variant of the confidentiality definition in general fails to correctly identify insecure channels. As an illustrative example, consider a blatantly insecure variant of TLS which outputs the encryption keys when given a ciphertext stream fragment c which contains two records of which the first can be authenticated correctly and the second fails the MAC verification (i.e., was adversarially modified). While such a construction should obviously be considered insecure, the $\mathsf{IND\text{-}CCFA}'$ definition fails to do so as, within the $\mathcal{O}'_{\mathsf{Recv}}$ oracle, the Recv algorithm is artificially called separately first on the genuine record and then on the modified one, thereby avoiding the scheme's vulnerability if the second record deviates from the first bit on.

We stress that, in contrast, the actual confidentiality notion IND-CCFA we defined for stream-based channels in Section 4.1 correctly identifies the described example above as insecure. This problem in the

[19]Most commonly, TLS implementations decode only one record at a time and output the resulting message, independently of whether there is an error message contained in a later packet in the queue or not.

$\mathsf{IND\text{-}CCFA}'$ case disappears if we assume fragment-independent behavior, as the split processing is now identical to the monolithic treatment.

Equivalence of confidentiality notions under fragmentation-independent behavior. We complete this section by showing that if one only considers channels with fragmentation-independent behavior, then the $\mathsf{IND\text{-}CCFA}$ notion from Definition 4.1 is actually equivalent to its split-call variant $\mathsf{IND\text{-}CCFA}'$. However, as the $\mathsf{IND\text{-}CCFA}$ notion also captures the security of channels which do not exhibit fragmentation-independent behavior (i.e., a larger class of channels), we conclude that the $\mathsf{IND\text{-}CCFA}$ option is a sound choice even for fragmentation-independent channels.

Theorem A.4 (Equivalence of $\mathsf{IND\text{-}CCFA}'$ and $\mathsf{IND\text{-}CCFA}$ under fragmentation-independent behavior)**.** *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a stream-based channel which exhibits fragmentation-independent behavior. Then* Ch *is* $\mathsf{IND\text{-}CCFA}'$*-secure if and only if it is* $\mathsf{IND\text{-}CCFA}$*-secure.*

Proof. It suffices to show that, assuming fragmentation-independent behavior of Ch, for every input ciphertext the oracles $\mathcal{O}_{\mathsf{Recv}}$ (from Figure 3) and $\mathcal{O}'_{\mathsf{Recv}}$ (from Figure 17) return the same output to $\mathcal{A}$ in a concrete execution. Note that for a synchronized receiver and valid prefix $C_R \| c \preccurlyeq C_S$ both oracles indeed output ϵ. Consider next any input ciphertext c and consider the case that synchronization is lost for c. Fix the following notation:

- In Figure 3, the $\mathsf{IND\text{-}CCFA}$ experiment: denote by m_1, respectively, m_2 the message fragments derived in lines 22, resp., 24 of oracle $\mathcal{O}_{\mathsf{Recv}}$ (i.e., $m_1 = \widetilde{m}$ and $m_2 = m \,\%\, [m, \widetilde{m}]$ in the experiment).
- In Figure 17, the $\mathsf{IND\text{-}CCFA}'$ experiment: denote by c'_1, resp., c'_2 the ciphertext fragments derived in lines 37, resp., 38 by $\mathcal{O}'_{\mathsf{Recv}}$ (i.e, $c'_1 = \widetilde{c}$ and $c'_2 = c \,\%\, \widetilde{c}$ in the experiment), and let m'_1 and m'_2 be the corresponding message fragments output by Recv.

Notice that by definition $m'_1 = m_1$ (indeed, the same pair of state st_R and ciphertext $\widetilde{c}$ are processed by Recv within $\mathcal{O}_{\mathsf{Recv}}$ and $\mathcal{O}'_{\mathsf{Recv}}$). However, since oracle $\mathcal{O}_{\mathsf{Recv}}$ returns message fragment m_2 to $\mathcal{A}$ while $\mathcal{O}'_{\mathsf{Recv}}$ returns m'_2, to prove the equivalence of the two notions we must also prove that $m'_2 = m_2$. For the assumed fragmentation-independent behavior of Ch it holds that $c'_1 \parallel c'_2 = c \implies m'_1 \parallel m'_2 = m$ and that $c_1 \preccurlyeq c \implies m_1 \preccurlyeq m$, hence $[m, m_1] = m_1$. Now it is immediate to derive the chain of equalities

$$m'_2 = m \,\%\, m'_1 = m \,\%\, m_1 = m \,\%\, [m, m_1] = m \,\%\, [m, \widetilde{m}] = m_2,$$

where the relations hold, from left to right, by fragmentation-independent behavior, by construction, by fragment-independent behavior, and (twice) by construction.

Once synchronization is lost ($\mathsf{sync} = 0$), note that both oracles will continue to output identical message fragments. This holds as both start at the same initial receiver state, and fragment-independence in particular then guarantees that $\| \mathbf{c} = \| \mathbf{c}'$ implies $\| \mathbf{m} = \| \mathbf{m}'$ (due to the symmetry in the definition). Since this even holds for invalid ciphertext sequences it must also be true here, when synchronization has already been lost. □

$\mathsf{IND\text{-}CCFA}$ **implies** $\mathsf{IND\text{-}CCFA}'$**.** While under fragmentation-independent behavior the $\mathsf{IND\text{-}CCFA}$ and $\mathsf{IND\text{-}CCFA}'$ notions are equivalent, we saw already that these notions are not equivalent (there exists schemes which are secure according to the $\mathsf{IND\text{-}CCFA}'$ but insecure according to the $\mathsf{IND\text{-}CCFA}$ notion). In what follows we will prove that the $\mathsf{IND\text{-}CCFA}$ notion is strictly stronger than the $\mathsf{IND\text{-}CCFA}'$ notion, that is, every scheme that is provably $\mathsf{IND\text{-}CCFA}$-secure is also $\mathsf{IND\text{-}CCFA}'$-secure.

Theorem A.5 ($\mathsf{IND\text{-}CCFA} \Longrightarrow \mathsf{IND\text{-}CCFA}'$)**.** *Let* Ch *be a stream-based channel. If* Ch *provides* $\mathsf{IND\text{-}CCFA}$ *security, then it also provides* $\mathsf{IND\text{-}CCFA}'$ *security. Formally, for every efficient* $\mathcal{A}$ *attacking the* $\mathsf{IND\text{-}CCFA}'$ *property there exists an efficient* $\mathcal{B}$ *attacking the* $\mathsf{IND\text{-}CCFA}$ *property such that*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}'}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCFA}}_{\mathsf{Ch},\mathcal{B}}.$$

Proof. Given a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ from the $\mathsf{IND\text{-}CCFA}$ game, to emulate the oracle $\mathcal{O}_{\mathsf{Recv}}$ algorithm $\mathcal{B}$ simply splits every queried fragment c that contains a non-empty genuine prefix into $\widetilde{c}$ and $c' = c \,\%\, \widetilde{c}$, ask $\mathcal{O}_{\mathsf{Recv}}$ for decryption of the two fragments in this order, hence give to $\mathcal{A}$ only the output of $\mathcal{O}_{\mathsf{Recv}}$ on input the second fragment c'. The effect of this pre-processing of $\mathcal{A}$'s queries is that $\mathcal{B}$'s receiving query will always be either genuine or fully deviating or ahead (i.e., the deviation starts from the very first bit). Thus, the oracle $\mathcal{O}_{\mathsf{Recv}}$ always processes these queries by executing either lines 11–13, or lines 14–17, or line 26, and hence it never has to perform the 'auxiliary call' to Recv, which does not occur in $\mathcal{O}'_{\mathsf{Recv}}$.

Beyond that, $\mathcal{B}$ forwards all other receiving queries to the $\mathcal{O}_{\mathsf{Recv}}$ oracle and all the sending queries to its left-or-right oracle $\mathcal{O}_{\mathsf{LoR}}$. Given this, it is immediate to see that $\mathcal{B}$'s simulation of the $\mathsf{IND\text{-}CCFA}'$ experiment is perfect, and hence

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCFA}'}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCFA}}_{\mathsf{Ch},\mathcal{B}}. \qquad \square$$

B IND-CCFA Attack Against Intuitively Confidential Scheme

Poettering [Poe16] in July 2016 pointed out a class of stream-based channels that are intuitively confidential but deemed insecure within our model, the construction $\mathsf{Ch}_{\mathsf{zeros}}$ from [Poe16] given in Figure 18 being an instance of that class. The $\mathsf{Ch}_{\mathsf{zeros}}$ construction uses an AEAD scheme as a building block and works as follows. The Send algorithm chops the input message fragment m into fixed-length blocks $m_1, \ldots, m_l$ (with $|m_i| = 1$ for simplicity), AEAD-encrypts these blocks sequentially using a running counter as associated data, and outputs the concatenation $c = c_1 \parallel \cdots \parallel c_l$ of the corresponding ciphertexts (which also have fixed length, $|c_i| = clen$). The Recv algorithm appends the input ciphertext fragment c to its buffer buf, extracts from the updated buffer the longest sequence of ciphertext blocks $c_1 \ldots c_l$ and keeps the remaining fragment in the buffer. It then AEAD-decrypts each block c_i and outputs the concatenation of the resulting message blocks $m_1 \parallel \cdots \parallel m_l$ as long as no AEAD decryption fails, otherwise it return an l-bit string of 0's.

By construction, Recv outputs non-error message bits (i.e., the fixed zero-string 0^l) on modified ciphertext input and hence $\mathsf{Ch}_{\mathsf{zeros}}$ provides no integrity protection. As this output is independent of the encrypted messages, this however should not harm confidentiality. Indeed, by confidentiality of the AEAD scheme, the left-or-right sending oracle should reveal no information about the challenge messages, while AEAD integrity should guarantee that the receiving oracle only returns a string of 0's when queried on out-of-sync ciphertext fragments. Thus, one would expect $\mathsf{Ch}_{\mathsf{zeros}}$ to provide confidentiality against chosen ciphertext-fragment attacks ($\mathsf{IND\text{-}CCFA}$).

The attack. The stream-based channel $\mathsf{Ch}_{\mathsf{zeros}}$ however is not confidential in the sense of $\mathsf{IND\text{-}CCFA}$. Consider an adversary $\mathcal{A}$ that proceeds as follows: it chooses $m_0 = 00$, $m_1 = 10$ and queries $c \leftarrow_{\$} \mathcal{O}_{\mathsf{LoR}}(m_0, m_1, 0)$. Let $c = c_1 c_2$ with $|c_1| = |c_2| = clen$. The adversary then derives $c' = c_1\overline{c_2}$ from c by inverting the bits of c_2 but leaving c_1 unmodified, and requests the decryption $m' \leftarrow \mathcal{O}_{\mathsf{Recv}}(c')$. Within the $\mathcal{O}_{\mathsf{Recv}}$ oracle we obtain $\mathsf{sync} = 0$, $\widetilde{c} = c_1$, and $\widetilde{m} = b$. Furthermore, due to integrity of the AEAD scheme, with high probability decrypting $\overline{c_2}$ yields an AEAD error $\perp$, and we thus have $m = 00$. Now, if $b = 0$, we have $[m, \widetilde{m}] = 0$ and thus $m' = 0$, and if $b = 1$ we have $[m, \widetilde{m}] = \varepsilon$ and thus $m' = 00$. The adversary outputs $b' = |m'| - 1$ and hence achieves a distinguishing advantage negligibly close to 1.

```
Init(1^λ):
  $K \leftarrow_\$ \mathcal{K}$
  $\mathsf{st}_{S,0} = (K, 0)$
  $\mathsf{st}_{R,0} = (K, 0, \varepsilon, 0)$
  return $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$

Send(st_S, m, f):
  parse $\mathsf{st}_S$ as $(K, \mathsf{seqno})$
  $l \leftarrow |m|$
  $m_1 \| \ldots \| m_l \leftarrow m$
      s.t. $|m_i| = 1$ for $i = 1, \ldots, l$
  $c \leftarrow \varepsilon$
  for $j \leftarrow 1$ to $l$ do
      $c_j \leftarrow \mathsf{Enc}(K, \mathsf{seqno}, m_j)$
      $c \leftarrow c \| c_j$
      $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
  $\mathsf{st}_S \leftarrow (K, \mathsf{seqno})$
  return $(\mathsf{st}_S, c)$

Recv(st_R, c):
  parse $\mathsf{st}_R$ as $(K, \mathsf{seqno}, \mathsf{buf}, \mathsf{fail})$
  $\mathsf{buf} \leftarrow \mathsf{buf} \| c$
  $l \leftarrow \lfloor |\mathsf{buf}| / clen \rfloor$
  $c_1 \| \ldots \| c_l \| \mathsf{buf} \leftarrow \mathsf{buf}$
      s.t. $|c_i| = clen$ for $i = 1, \ldots, l$
  $m \leftarrow \varepsilon$
  for $j \leftarrow 1$ to $l$ do
      $m_j \leftarrow \mathsf{Dec}(K, \mathsf{seqno}, c_j)$
      if $m_j = \bot$ then
          $\mathsf{fail} \leftarrow 1$
      $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
  if $\mathsf{fail} = 1$ then
      $m \leftarrow 0^l$
  $\mathsf{st}_R \leftarrow (K, \mathsf{seqno}, \mathsf{buf}, \mathsf{fail})$
  return $(\mathsf{st}_R, m)$
```

Figure 18: A stream-based channel $\mathsf{Ch}_{\mathsf{zeros}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ that uses an AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ as a building block. We assume the length of ciphertexts output by Enc on 1-bit messages to be constant, $clen$ bits. As we explain in Appendix B, $\mathsf{Ch}_{\mathsf{zeros}}$ is an intuitively confidential channel but is declared insecure according to our IND-CCFA notion.

Discussion. The receiving algorithm of channel $\mathsf{Ch}_{\mathsf{zeros}}$ returns a valid message fragment even if the AEAD decryption internally rejects, in which case it outputs a string of 0's. On the one hand, this channel construction is artificial as it explicitly hides decryption errors, neglecting integrity. On the other hand, it does constitute a reasonable scheme in a setting where only confidentiality matters and should be treated as such. This exemplifies that our IND-CCFA notion excludes some schemes that are intuitively confidential and hence might be considered too strong.

Let us see which part of the IND-CCFA definition is responsible for this. As discussed, AEAD security should ensure that the decryption m of the out-of-sync ciphertext $c' = c_1\overline{c_2}$ reveals nothing about the challenge message m_b. However, within the IND-CCFA experiment, when answering query c' the receiving oracle does leak information about m_b: it reveals whether $m = 00$ has a non-empty common prefix with $\widetilde{m} = b$ or, equivalently, whether $b = 0$ or not. So, while m does not yield any information about m_b, the oracle $\mathcal{O}_{\mathsf{Recv}}$ artificially does. It thereby enables an attack that intuitively would not be feasible in practice.

C INT-CST Implies INT-PST

Proposition C.1. *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a correct stream-based channel which is* INT-CST *secure. Then the channel is also* INT-PST *secure. Furthermore,* $\mathsf{Adv}^{\mathsf{INT\text{-}PST}}_{\mathsf{Ch},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{INT\text{-}CST}}_{\mathsf{Ch},\mathcal{A}}(\lambda)$ *for any algorithm* $\mathcal{A}$.

Proof. Assume that $\mathcal{A}$ attacks the INT-PST property of the channel. First note that $\mathcal{A}$ has the same interfaces as if attacking INT-CST such that we can think of running both experiments simultaneously. It then suffices to show that, if we set $\mathsf{win} \leftarrow 1$ in Line 14 of the INT-PST experiment, then we would also set $\mathsf{win} \leftarrow 1$ (in Line 18 or 35) in the simultaneous execution of the INT-CST experiment (both in Figure 5). Note that as long as $\mathsf{sync} = 1$ and the ciphertext stream submitted to $\mathcal{O}_{\mathsf{Recv}}$ is a prefix of the one created by $\mathcal{O}_{\mathsf{Send}}$, then the receiver's oracle in experiment INT-CST would indeed return the recovered message fragment, as in the INT-PST experiment.

Suppose that $\mathcal{A}$ triggers $\mathsf{win} \leftarrow 1$ in the INT-PST experiment. If at this point $C_R \preccurlyeq C_S$ then, because of correctness of the channel, we must also have $M_R \preccurlyeq M_S$, implying that win would not have been set. It follows that there must exist some $C_R \parallel c \not\preccurlyeq C_S$ where c is the first call to $\mathcal{O}_{\mathsf{Recv}}$ where the concatenation of the receiver strings deviate from or exceed the ciphertext stream of the sender. At this point we must also have $\mathsf{sync} = 1$ and $\mathsf{win} = 0$ in the $\mathsf{INT\text{-}CST}$ experiment and we enter the third case in Line 22.

If in this case c contains some non-deviating prefix $\widetilde{c} \preccurlyeq c$, we compute $\widetilde{c}$ such that $C_R \parallel \widetilde{c} \preccurlyeq C_S$ is maximal. It again follows from correctness that for the processed $\widetilde{c}$ we get a message fragment $\widetilde{m}$ such that $M_R \parallel \widetilde{m} \preccurlyeq M_S$. But in order to set $\mathsf{win} \leftarrow 1$ in the $\mathsf{INT\text{-}PST}$ experiment, we must have that the full fragment c makes Recv output a message fragment m such that m contains message bits beyond the common prefix with M_S. It follows that $m' \leftarrow m \% [m, \widetilde{m}]$, the fragment of m beyond $\widetilde{m}$, too, must contain some non-trivial entries, different from error symbols. But then we also set $\mathsf{win} \leftarrow 1$ in the $\mathsf{INT\text{-}CST}$ experiment run.

If c contains only deviating (or exceeding) bits or in the case that synchronization was lost before, the full resulting message fragment (m' resp. m) is considered for the winning condition, i.e., whenever $\mathcal{A}$ in this case wins in the $\mathsf{INT\text{-}PST}$ experiment, it also does in the $\mathsf{INT\text{-}CST}$ experiment. □

D aINT-CST Implies aINT-PTXT

Proposition D.1 ($\mathsf{aINT\text{-}CST} \implies \mathsf{aINT\text{-}PTXT}$)**.** *Let* $\mathsf{aCh} = (\mathsf{aInit}, \mathsf{aSend}, \mathsf{aRecv})$ *be a correct atomic-message channel. If* aCh *provides integrity of ciphertext streams then it also provides integrity of plaintexts and, in particular,* $\mathsf{Adv}^{\mathsf{aINT\text{-}PTXT}}_{\mathsf{aCh},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{aINT\text{-}CST}}_{\mathsf{aCh},\mathcal{A}}(\lambda)$ *for any adversary* $\mathcal{A}$.

Proof. Consider an execution of the $\mathsf{aINT\text{-}PTXT}$ experiment with an adversary $\mathcal{A}$ against the channel aCh. Since the $\mathsf{aINT\text{-}PTXT}$ and $\mathsf{aINT\text{-}CST}$ experiments both provide interfaces to a sending oracle $\mathcal{O}_{\mathsf{Send}}$ and a receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ we can imagine to run the two experiments simultaneously and show that if $\mathcal{A}$ is successful in the former, so is in the latter. More specifically, we show that if the $\mathsf{aINT\text{-}PTXT}$ experiment sets $\mathsf{win} \leftarrow 1$ in line 16 then the $\mathsf{aINT\text{-}CST}$ experiment sets $\mathsf{win} \leftarrow 1$ in lines 23 or 34.

Observe that $\mathcal{A}$ triggers the execution of line 16 (setting win to 1) if it submits some ciphertext fragments to $\mathcal{O}_{\mathsf{Recv}}$ in experiment $\mathsf{aINT\text{-}PTXT}$ that result in a message sequence $\mathbf{M_R}$ deviating from the genuine message sequence $\mathbf{M_S}$, beyond errors. By correctness, a deviation in the message sequence can only originate from a deviation in the ciphertext sequence and, thus, we know that $\mathcal{A}$ submits at least one out-of-sync ciphertext. Suppose that $\mathcal{A}$ triggers the execution of line 16 of the $\mathsf{aINT\text{-}PTXT}$ experiment already when it submits the first out-of-sync ciphertext: then such ciphertext causes the execution of instruction 28 and the following loop in the $\mathsf{aINT\text{-}CST}$ experiment, yielding a vector $\mathbf{m}'$ that contains all messages received so far but the longest common prefix with the sequence of sent messages. It follows from the assumptions made that $\mathbf{m}'$ contains some valid message and, thus, $\mathsf{win} \leftarrow 1$ is set in line 34 in the $\mathsf{aINT\text{-}CST}$ experiment, too. Suppose now that $\mathcal{A}$ enforces the execution of line 16 of the $\mathsf{aINT\text{-}PTXT}$ experiment after the first out-of-sync ciphertext has been submitted. Then we have two possibilities: either the deviating part of the first out-of-sync ciphertext fragment produces some valid messages and, thus, in the $\mathsf{aINT\text{-}CST}$ experiment sets $\mathsf{win} \leftarrow 1$ in line 34, or some of the following ciphertext fragments that $\mathcal{A}$ submits cause aRecv to output valid messages, hence causing $\mathsf{aINT\text{-}CST}$ to set $\mathsf{win} \leftarrow 1$ in line 23. □