Data Is a Stream: Security of Stream-Based Channels

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

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

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 finally 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.

Keywords secure channel, data stream, AEAD, confidentiality, integrity

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 [1] mobile telecommunications systems, in WEP, WPA and WPA2 [19] (which secure wireless LAN communications), IPsec [22] (which provides security at the IP layer), TLS [15] and DTLS [31] (which run over TCP [30] and UDP [29], respectively), Google's QUIC protocol [33], and SSH [36] (an 'application layer' secure protocol).

AEAD and secure channels in the literature. Authenticated Encryption with Associated Data (AEAD) [32] 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 is insufficient for constructing secure channels. 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 when 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. [5] 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 Black et al. [23] to include a richer variety of features, suitable for handling channels that permit (or deny) replays, message drops, and reordering. Additional literature concerning the formalization of secure channels includes [34,12,13,26,25,27,20,24,3].

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. 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 [17]) often attempt to use stream-oriented secure channels (like TLS) to perform secure, atomic message delivery.

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. [2]. 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 [5] 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 [14]. 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 [35,7]. All these attacks show the incompleteness of previous approaches to modeling and analyzing secure channels.

Boldyreva et al. [9] extended the classical, atomic secure channel notions to cover the case of SSH-like stream-based secure channels, broadening the SSH-specific work of [28]. However, while they allow for fragmented delivery of ciphertexts to the receiver, their work still assumes that the encryption process on the sender's side is atomic, meaning that there is a one-to-one correspondence between message 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 channels protocols. For example, even though the TLS specification [15] 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), and, 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*. Our models are in the game-based tradition, and extend those of [5,9] to handle the streaming nature of the channels that we consider.

While our methodology and modeling closely resemble those of [9], and indeed build upon them, a crucial difference comes in our treatment of the sending (or encrypting) function of a stream-based channel: in [9], 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 [9]. In addition, we de-

velop suitable integrity notions for the streaming setting, whereas [9] does not consider this aspect. This is important because the (informal) security properties that applications expect a secure channel to provide include confidentiality as well as integrity, while security in the most powerful 'chosen fragment attack' setting of [9] does not provide *any* integrity guarantees.

Bringing integrity into the picture for stream-based channels also enables us to prove a composition result analogous to the classical result of [6] 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 appropriate 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 [10] 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 [6] does not go through. In [10], this was overcome by assuming the scheme is such that 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 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. So our security results provide validation for this important real-world protocol design, whilst fully taking its streaming behavior into account. In the full version of this paper we moreover propose a generic construction of a stream-based channel from symmetric encryption supporting fragmentation as per [9].

Also in the full version, we return to the starting point of our discussion and analyze how applications can use stream-based channels to safely transport atomic messages by encoding distinguished end-of-message symbols into the sent message stream to identify the atomic messages' boundaries. Establishing the security of this simple and natural approach however requires the introduction of an additional technical property orthogonal to integrity and confidentiality. Our analysis sheds a new formal light on the truncation [35] and 'cookie-cutter' [7] 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.

Further related work. Bhargavan et al. [8] 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).

A seemingly similar line of work to ours concerns blockwise-adaptive security and on-line symmetric encryption schemes, as developed in [4,21,18,11]. 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. 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.

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. Finally, in Section 5 we show feasibility of our notions by providing a generic construction of a stream-based channel. We conclude with open questions arising from this work in Section 6.

2 Preliminaries

Notation. Let Σ be an alphabet and $s \in \Sigma^*$. We indicate by $|s|$ the length of s, by $s[i]$ its i-th character, and by $s[i,\ldots,j]$ the substring $s[i]||\ldots||s[j]$, where $||$ denotes the string concatenation. Let $s, t \in \Sigma^*$. We say that s is a *prefix* of t and write $s \prec t$ if there exists $r \in \Sigma^*$ such that $s||r = t$; in this case we write $r = t \,\%\, s$. We denote the longest common prefix of s and t by $[s,t] = [t,s]$. Note that $s \prec t$ if and only if $[s,t] = s$. Using the above notation we will often consider $s \,\%\, [s,t]$, i.e., the suffix of s with the longest common prefix of s and t stripped off. Let $\boldsymbol{s} = (s_1,\ldots,s_\ell) \in (\Sigma^*)^\ell$ be a vector of strings for some integer ℓ; if $\boldsymbol{s}$ is empty, i.e., $\ell = 0$, we denote this by $\boldsymbol{s} = ()$. For every $0 \le i \le j \le \ell$ we denote $\boldsymbol{s}[i] = s_i$ and $\boldsymbol{s}[i,\ldots,j] = (s_i,\ldots,s_j)$; we use the shortcut $||\boldsymbol{s}$ for the concatenation $s_1||\ldots||s_\ell$, and conventionally define $||() = \varepsilon$. We say that two vectors $\boldsymbol{s} = (s_1,\ldots,s_\ell)$ and $\boldsymbol{t} = (t_1,\ldots,t_{\ell'})$ are equal and write $\boldsymbol{s} = \boldsymbol{t}$ if and only if $\ell = \ell'$ and $s[i] = t[i]$ for all $1 \le i \le \ell$. Slightly overloading notation, we denote the merge of two vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ as $\boldsymbol{s}||\boldsymbol{t} = (s_1,\ldots,s_\ell,t_1,\ldots,t_{\ell'})$.

Channel terminology. Our syntax for channels is intentionally independent of the targeted security properties as these may vary from one specific application to another. To reflect the generic functionality of channels and maintain a higher level of abstraction than, e.g., in the case of authenticated encryption, we define sending (Send) and receiving (Recv) 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) [30] and, in

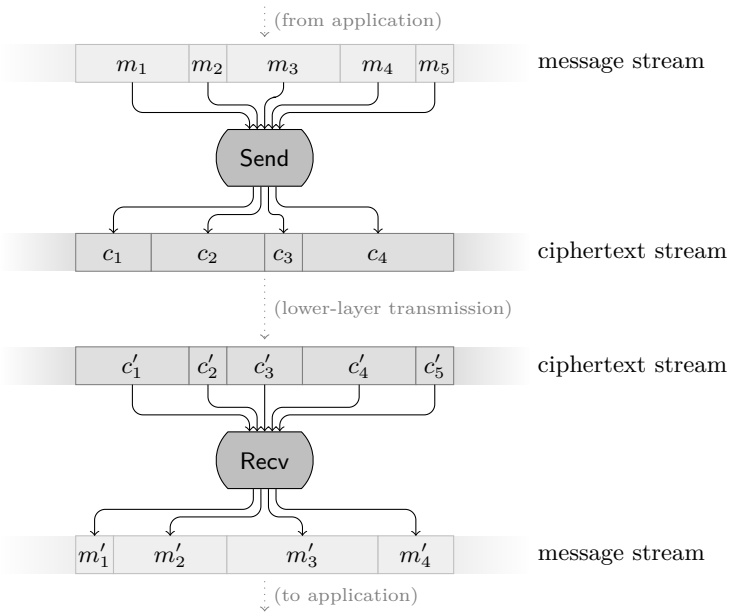


Fig. 1: Illustration of the behavior of the Send and Recv algorithms of a stream-based channel, indicating the message and ciphertext fragments being sent (m_i resp. c_i) and received (m'_i resp. c'_i).

a second step, we define confidentiality and integrity properties expected from (stream-based) secure channel protocols like the Transport Layer Security (TLS) Record Protocol [15] or the Secure Shell (SSH) Binary Packet Protocol [37].[1] To do so we first need to define the syntax of stream-based channels that, in contrast to previous models for channel, 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 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.

[1] 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) [29] as done, e.g., in Google's Quick UDP Internet Connections (QUIC) protocol [33], 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 in an isolated manner (e.g., using an AEAD scheme).

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. 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 fragmentation. 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. In our definition below we demand that each message fragment m_i processed by Send results in a ciphertext fragment c_i. Since a ciphertext fragments can be empty ($c_i = \varepsilon$), this implicitly enables Send to wait for more data by outputting empty ciphertext fragments. Figure 1 illustrates the behavior of the sending and receiving algorithms of a stream-based channel.

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

Definition 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 $\mathcal{E}$ consists of three efficient probabilistic algorithms:*

- Init. *On input of a security parameter 1^λ, this 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 of a state $\mathsf{st}_S \in \mathcal{S}_S$, a fragment $m \in \{0,1\}^*$, and a flush flag $f \in \{0,1\}$, this 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 of a state $\mathsf{st}_R \in \mathcal{S}_R$ and a ciphertext fragment $c \in \{0,1\}^*$, this 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 $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in (\{0,1\}^*)^\ell$ and of flush flags $\boldsymbol{f} = (f_1, \ldots, f_\ell) \in \{0,1\}^\ell$, let $(\mathsf{st}_S, \boldsymbol{c}) \leftarrow_\$ \mathsf{Send}(\mathsf{st}_{S,0}, \boldsymbol{m}, \boldsymbol{f})$ 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 $\boldsymbol{c} = (c_1, \ldots, c_\ell)$ and $\mathsf{st}_S = \mathsf{st}_{S,\ell}$. For $\ell = 0$ we define $\boldsymbol{c}$ 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.

Intuitively, correctness of stream-based channels guarantees that for every message fragments input to Send, if the corresponding ciphertext stream is processed by Recv, then no matter how the ciphertext stream is (re)fragmented at the receiver side the returned message stream is a prefix of the initial message stream. Moreover, when Recv consumes a ciphertext fragment generated by a call to Send with the flush flag set to 1, its output stream contains all the message fragments input to Send up to that call. We next formalize this intuition.

Definition 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 pair* $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow_{\$} \mathsf{Init}(1^{\lambda})$*, all* $\ell, \ell' \geq 0$*, all choices of the randomness for algorithms* $\mathsf{Init}, \mathsf{Send}$ *and* Recv*, all message-fragment vectors* $\boldsymbol{m} \in (\{0,1\}^*)^{\ell}$*, all flush-flag vectors* $\boldsymbol{f} \in \{0,1\}^{\ell}$*, all sending output sequences* $(\mathsf{st}_{S,\ell}, \boldsymbol{c}) \leftarrow_{\$} \mathsf{Send}(\mathsf{st}_{S,0}, \boldsymbol{m}, \boldsymbol{f})$*, all ciphertext-fragment vectors* $\boldsymbol{c'} \in (\{0,1\}^*)^{\ell'}$*, and all receiving output sequences* $(\mathsf{st}'_{R,\ell'}, \boldsymbol{m'}) \leftarrow_{\$} \mathsf{Recv}(\mathsf{st}_{R,0}, \boldsymbol{c'})$*, we have*

$$||\boldsymbol{c} = ||\boldsymbol{c'} \implies ||\boldsymbol{m}[1,\ldots,i] \prec ||\boldsymbol{m'} \prec ||\boldsymbol{m},$$

where $i = \max(\{0\} \cup \{j : f_j = 1\})$ *is the largest index such that the flush flag* $f_i = 1$ *(i.e., if all flush flags are set to zero then* $i = 0$ *and* $\boldsymbol{m}[1,\ldots,i] = \varepsilon$*).*

Remark 1. Correctness implies that if we feed Recv with a prefix of the ciphertext stream output by Send, i.e., $||\boldsymbol{c'} \prec ||\boldsymbol{c}$, then the receiver outputs a prefix of the corresponding message stream, $||\boldsymbol{m'} \prec ||\boldsymbol{m}$, since

$$||\boldsymbol{c'} \prec ||\boldsymbol{c} \Rightarrow \exists\, c'' \in \{0,1\}^* : ||\boldsymbol{c'}\,||\,c'' = ||\boldsymbol{c} \overset{\text{(corr.)}}{\Longrightarrow} ||\boldsymbol{m'}\,||\,m'' \prec ||\boldsymbol{m} \Rightarrow ||\boldsymbol{m'} \prec ||\boldsymbol{m}$$

for all receiving output sequences $(\mathsf{st}'_{R,\ell'+1}, m'') \leftarrow_{\$} \mathsf{Recv}(\mathsf{st}'_{R,\ell'}, c'')$.

Remark 2. It is instructive to compare our correctness definition with that of Boldyreva et al. [9]. There, correctness requires that if a sequence $\boldsymbol{m}$ of discrete messages is encrypted, and the resulting ciphertext stream $||\boldsymbol{c}$ is then decrypted (possibly in a fragmented manner), then the obtained message sequence (when message separators ¶ are removed) is identical to the original sequence $\boldsymbol{m}$. In the special case of a single message, this implies that encryption 'always flushes' in the setting of [9], 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 later sending. This is, then, an essential difference between the setting of Boldyreva et al. [9] and the streaming one. An additional difference is that the correctness condition in [9] is stronger than ours as it incorporates a certain amount of robustness. More specifically, the sequence of ciphertext fragments $\boldsymbol{c'}$ submitted for decryption in the correctness definition of [9] may extend the sequence produced by encryption (in other words, $||\boldsymbol{c}$ is only required to be a prefix of $||\boldsymbol{c'}$ for decryption to still work correctly up to $||\boldsymbol{c}$).

4 Security for Stream-Based Channels

In the following we introduce both confidentiality and integrity notions attuned to the stream-based setting and analyze their composition. We provide corresponding notions in terms of asymptotic security; analogous notions in the concrete setting are easy to infer.[2]

[2] 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.

4.1 Confidentiality

As in the ciphertext fragmentation setting introduced by Boldyreva et al. [9], whose confidentiality notion in turn is inspired by the IND-sfCCA notion by Bellare et al. [5], our security notions have to deal with the fact that stream-based channels support processing of arbitrary fragments of the message resp. ciphertext stream. While Boldyreva et al. [9] considered only fragmented decryption (but atomic encryption) and therefore focused their attention on the CCA-like setting, the fragmented message processing of stream-based channels in our case also affects the adversarial capabilities in the CPA-like setting. We hence define security notions both for the case of *chosen plaintext-fragment* attacks (IND-CPFA) as well as *chosen ciphertext-fragment* attacks (IND-CCFA).

Adapting the chosen-plaintext capabilities of an adversary to the stream-based settings is relatively straightforward (incorporating the standard left-or-right oracle). However, deriving a sound security notion for an adversary controlling the fragmentation on the received ciphertext stream turns out to be more delicate. In general, chosen-ciphertext-like oracles strive to allow decryption of as much of the input as possible without enabling trivial attacks. We follow the approach of Bellare et al. [5] to model stateful (decryption) security notions by considering the receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ to be in-sync and not returning a response to the adversary $\mathcal{A}$ as long as $\mathcal{A}$ supplies (parts of) the original ciphertext stream output by the left-or-right sending oracle $\mathcal{O}_{\mathsf{LoR}}$ in correct sequential order. When $\mathcal{A}$ deviates from the original ciphertext stream, the $\mathcal{O}_{\mathsf{Recv}}$ oracle is considered out of sync and, from that point on, the output of the Recv algorithm is given to the adversary.

For a sound definition we are faced with the question: At which point *exactly* shall $\mathcal{O}_{\mathsf{Recv}}$ be considered out-of-sync? Boldyreva et al. decided to stay close to the original definitions of Bellare et al. and conservatively defined synchronization to be lost at ciphertext boundaries (i.e., their notion reveals the decryption of the full ciphertext as output by Send whenever any part of it is modified). However this option is inappropriate in our stream-based setting where the output of Send is not necessarily an atomic unit.

As an example to illustrate this, 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 payload in a single TLS record, Send is forced to output a ciphertext fragment which contains (at least) two TLS records. An adversary which now forwards this fragment to the decryption oracle in the IND-sfCFA definition of Boldyreva et al. [9, Definition 4] with the second record modified but the first record untouched will be provided with the decryption of *both* records, thereby trivially revealing parts of the challenge message string.

Mindful of this example and taking into account that the output of Send in our case is a bit stream without any further structure in general, the natural choice appears to consider $\mathcal{O}_{\mathsf{Recv}}$ to become out-of-sync exactly when the first bit of its ciphertext stream input deviates from the genuine output of Send.

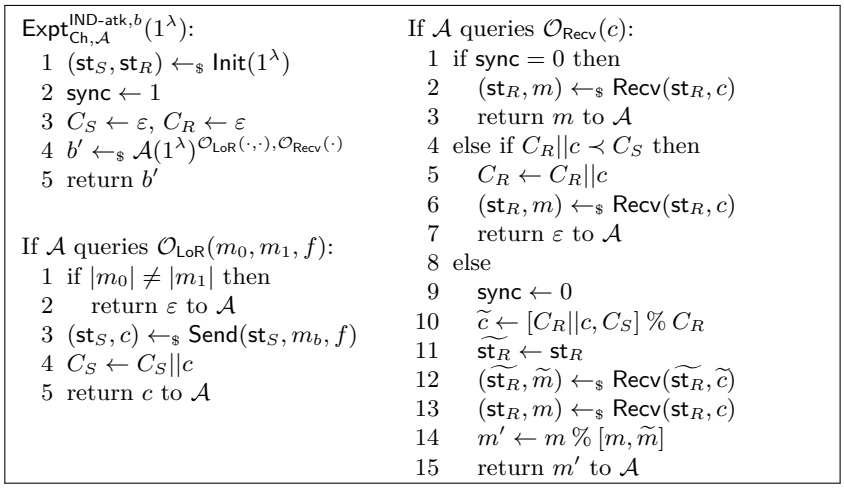


Fig. 2: Security experiment for *confidentiality* (IND-atk) of stream-based channels. A CPFA-attacker only has access to the oracle $\mathcal{O}_{\mathsf{LoR}}$.

In more detail, we define our stream-based confidentiality notions IND-CPFA (indistinguishability under chosen plaintext-fragment attack) and IND-CCFA (indistinguishability under chosen ciphertext-fragment attack) through the experiment $\mathsf{Expt}^{\mathsf{IND\text{-}atk},b}_{\mathsf{Ch},\mathcal{A}}$ (where atk is a placeholder for either CPFA or CCFA), depicted in Figure 2. The adversary's goal in the experiment $\mathsf{Expt}^{\mathsf{IND\text{-}atk},b}_{\mathsf{Ch},\mathcal{A}}$ is to guess the bit b. In the experiment the $\mathcal{O}_{\mathsf{LoR}}$ oracle provides the adversary with the response of Send to the (left or right) message fragment input. The oracle first checks if the input message fragments m_0 and m_1 have the same bit length (i.e., $|m_0| = |m_1|$). If this is the case, it invokes Send on m_b, adds its response c to the internal ciphertext stream variable C_S and provides $\mathcal{A}$ with c.

The $\mathcal{O}_{\mathsf{Recv}}$ oracle in the experiment processes the ciphertext fragment input (thereby updating the receiving state st_R), but artificially suppresses the output of Recv as long as the fragments are in sync. In case synchronization has been already lost (i.e., $\mathsf{sync} = 0$), $\mathcal{O}_{\mathsf{Recv}}$ simply passes the output of Recv to $\mathcal{A}$. Otherwise, it checks whether the concatenation C_R of ciphertext fragments seen so far together with the current fragment c is still a prefix of the ciphertext stream C_S output by $\mathcal{O}_{\mathsf{LoR}}$: if this is the case, Recv is invoked on c but its output is suppressed. Otherwise $\mathcal{O}_{\mathsf{Recv}}$ is now considered out-of-sync and there are two definitional options available, both following the paradigm of giving as much information to the adversary as possible without enabling trivial attacks: The first option is to split the call to the receiver into two, one for the longest common prefix $\widetilde{c}$ of the received ciphertext c which still matches the ciphertext stream C_S output by $\mathcal{O}_{\mathsf{LoR}}$, and one for the remaining ciphertext part where they diverge. The second option, and this is the one we use here and which turns out to be more appropriate than the first one (as we discuss in the full version), is to

run the receiver on the full ciphertext c and later suppress parts of the message stream which the receiver *would* have obtained when run on $\widetilde{c}$.

More formally, our suppression strategy on the level of the message stream first simulates a Recv call on a copy of the current state st_R and $\widetilde{c}$ and registers its output $\widetilde{m}$. Second, Recv is regularly invoked (again for the original state st_R) on the full ciphertext fragment c provided by the adversary, resulting in a message m being output. Finally, the common prefix of m and $\widetilde{m}$ (i.e., any potential challenge message stream bits in m) is suppressed and the remaining part of m is passed to $\mathcal{A}$.

Definition 3 ($\mathsf{IND\text{-}CPFA}$ 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 2, 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 (resp. 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:*

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

For the sake of completeness we comment on the alternative, intuitively appealing way for defining the receiving oracle by splitting the ciphertext in our setting in in the full version, which however leads to a confidentiality notion that only covers a smaller class of channels.

4.2 Integrity

In this section 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. [9] only addressed confidentiality in the presence of ciphertext fragmentation. 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 [5] 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. 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

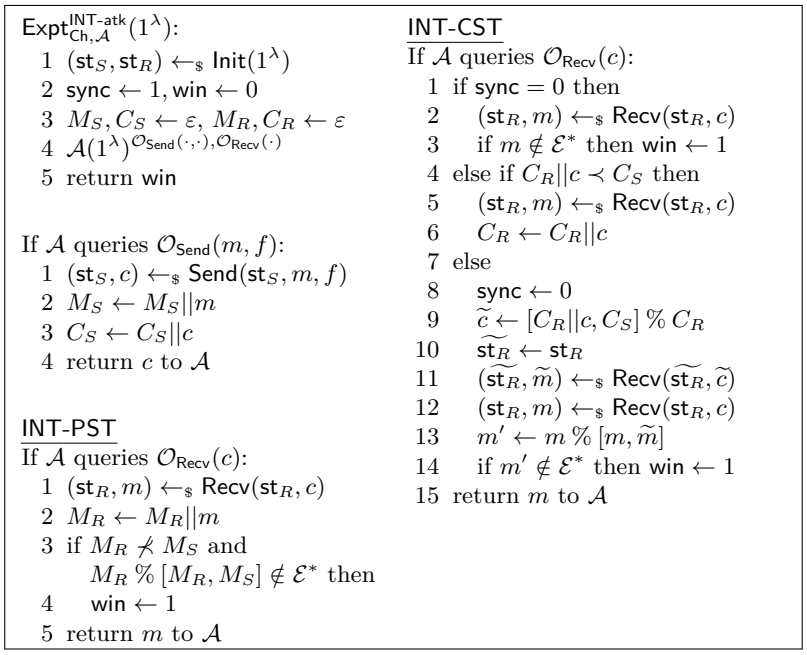


Fig. 3: Security experiment for *integrity* (INT-atk) of stream-based channels. An PST-attacker is provided with access to the middle $\mathcal{O}_{\mathsf{Recv}}$ oracle (INT-PST), whereas a CST-attacker is instead granted access to the oracle on the right-hand side (INT-CST).

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 INT-sfCTXT definition of [5] would allow trivial attacks by declaring successful any adversary that makes the Recv buffer (part of) an out-of-sync ciphertext. 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 nonemtpy message fragment resulting from an out-of-sync 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 3. 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 INT-PST and INT-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 INT-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 INT-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. In the moment when an out-of-sync ciphertext 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.

Definition 4 (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 2, 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 *(resp.* 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 3. Our definitions of integrity do not preclude from being secure those channels 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 attaching importance to ciphertext boundaries. Hence, this is quite distinct from the usual 'atomic' setting. In particular, applications that use a streaming channel to transmit atomic messages must take extra care to ensure no partially retrieved message fragment from the streaming channel is processed as if it was a complete (atomic) message, as such misinterpretation can lead—and in the past has led—to attacks [35,7].

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 of 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. 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 [9] which, 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

Due to space restrictions we comprehensively discuss the relations among the introduced security notions for the streaming setting only in the full version. In short, we show that, for both confidentiality and integrity, the stronger notion implies the weaker one, i.e., IND-CCFA $\Rightarrow$ IND-CPFA and INT-CST $\Rightarrow$ INT-PST, as one might expect. Further, we extend the composition result from [6]—that (stateful) IND-CPA and INT-CTXT together imply (stateful) IND-CCA—to our streaming setting. Interestingly, the analogous prerequisites IND-CPFA and INT-CST alone are not sufficient to establish the composition result in our case: we additionally require the channel to be *error predictable* (ERR-PRE). The latter notion, defined only in the full version due to space restrictions, formalizes the ability to efficiently predict the error messages that should be obtained when the receiving algorithm fails.

Error predictability assists the security proof for our composition theorem in two ways. First, it allows us to deal with the problem of having multiple decryption errors [10]. This problem also appears in the atomic setting and has been surmounted there by considering only single error messages [6] or by restricting the likelihood of different error messages to appear [10]. Our notion of error predictability gives a more general approach which is also applicable in the atomic setting. Secondly, error predictability directly supports the reduction to the integrity property INT-CST in our proof. In our stream-based scenario we basically must be able to tell if the receiver is still buffering ciphertext fragments, or if it can already produce an error message. Error predictability gives us exactly this.

We stress, and will expand in Section 5, that error predictability can be met by natural constructions. The composition result for stream-based channels is summarized in the theorem below. We provide a formal proof of this result in the full version.

Theorem 1 (INT-CST $\wedge$ IND-CPFA $\wedge$ ERR-PRE $\Rightarrow$ IND-CCFA)**.** *Let* Ch = (Init, Send, Recv) *be a (correct) stream-based channel with associated error space* $\mathcal{E}$*. If* Ch *provides integrity of ciphertext streams, error predictability, and indistinguishability under chosen plaintext-fragment attacks then it also provides indistinguishability under chosen ciphertext-fragment attacks. Formally, for ev-*

ery 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*

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

5 Construction of Stream-Based Channels

In this section we demonstrate the feasibility of our security notions by providing a generic construction of stream-based channels which directly bases on the well-established primitive of authenticated encryption with associated data and provides strong security in terms of confidentiality as well as integrity. Although it is rather illustrative than definitive, we remark that our construction is quite close to the TLS Record Protocol.

We define the generic construction of a stream-based channel $\mathsf{Ch}_{\mathsf{AEAD}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ based on an authenticated encryption with associated data (AEAD) scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and distinguished error symbol $\perp$ as introduced by Rogaway [32].[3] The encryption algorithm $\mathsf{Enc}\colon \mathcal{K} \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ on input a key, an associated data string, and a message, outputs a ciphertext. The decryption algorithm $\mathsf{Dec}\colon \mathcal{K} \times \{0,1\}^* \times \{0,1\}^* \to (\{0,1\}^* \cup \{\perp\})$ on input a key, an associated data string, and a ciphertext, outputs either a message or the distinguished error symbol. We assume that the AEAD scheme allows the encryption of variable-length messages of up to il bits and that the ciphertext output for such messages has length at most $2^{\mathsf{ol}} - 1$ bits. This enables us to encode the length of ciphertexts with a fixed-size string of ol bits.

Our channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ is displayed in Figure 4 and 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 $\mathcal{E} = \{\perp\}$. The channel works as follows.

- The Init 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, initially set to 0.
- The Send algorithm keeps on buffering input message strings until it has collected at least il bits. If sufficiently many bits have been collected, then Send encrypts message chunks m' of length il bits using the AEAD scheme on input message m' and associated data a running sequence number seqno.[4] The ciphertext generated is then prepended with the binary encoding of its size

[3] Although our construction does not incorporate nonces it can easily be extended to the nonce-based setting as originally defined by Rogaway [32].

[4] A more natural construction in the nonce-based setting would use seqno as the encryption nonce and have empty associated data input. We have chosen the current construction because of its closeness to TLS, which treats its sequence number as associated data.

(with the fixed number of ol bits) and the result appended to the ciphertext string c to be output. Note that the size encoding is not authenticated. In case the Send algorithm was called with the flush flag set to 1, in a final step it also encrypts any remaining buffered message in the same way, in order to empty the message buffer (this message will potentially be of length smaller than il).
- The Recv algorithm outputs an error (without any further state modification) once a first error has emerged from the AEAD decryption algorithm in some previous call; 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 the complete AEAD ciphertext of the indicated length and, if so, strips it from the buffer, decrypts it (incrementing the sequence number used in the associated data), and appends the result to the message to be output. This process is repeated until there is no completely parsable ciphertext left. However, in case the AEAD decryption algorithms outputs an error, after appending this error symbol to the output message, the Recv algorithm sets the failure flag fail to 1 and stops parsing further input.

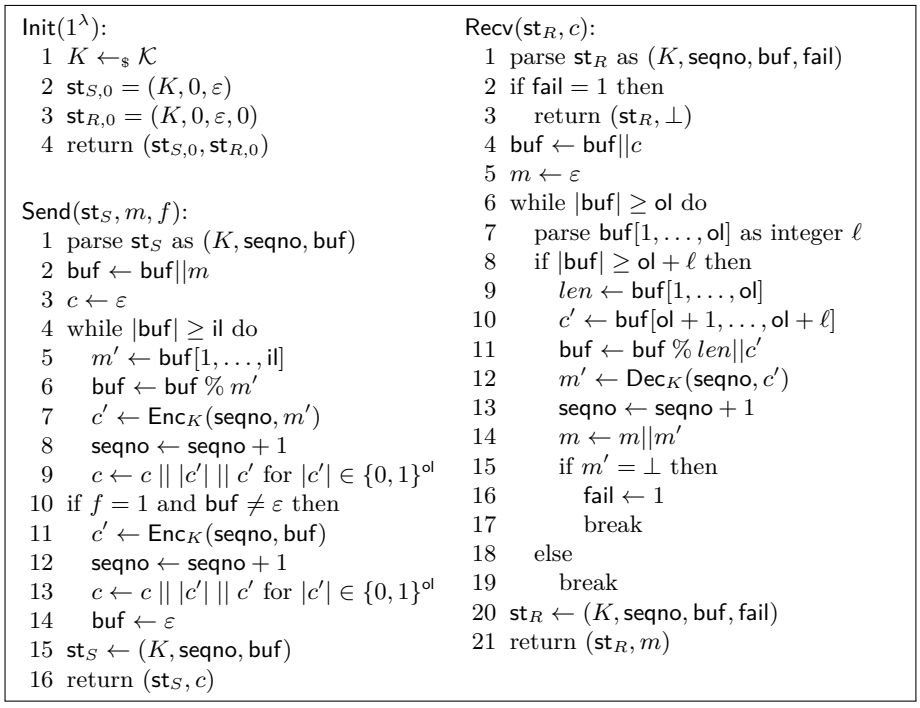
Correctness of $\mathsf{Ch}_{\mathsf{AEAD}}$ follows from the correctness of the AEAD scheme.

Security analysis. Our generic stream-based channel construction $\mathsf{Ch}_{\mathsf{AEAD}}$ from Figure 4 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 by Rogaway [32].[5] Using Theorem 1 we can moreover infer that it also provides indistinguishability under chosen ciphertext-fragment attacks (IND-CCFA). We provide the detailed security analysis in the full version of this paper.

5.1 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 of a stream-based channel based on authenticated encryption with associated data is actually very close to the TLS Record Protocol when using an AEAD scheme as specified for TLS version 1.2 [15, Section 6.2.3.3] and in the current draft for TLS version 1.3 [16, Section 6.2.2]: 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

[5] Note that Rogaway [32] actually defines the stronger IND$-CPA notion which implies IND-CPA security based on a standard left-or-right encryption oracle. We only require IND-CPA though as it is sufficient for our security proof.



$\mathsf{Init}(1^\lambda)$:
1 $K \leftarrow_\$ \mathcal{K}$
2 $\mathsf{st}_{S,0} = (K, 0, \varepsilon)$
3 $\mathsf{st}_{R,0} = (K, 0, \varepsilon, 0)$
4 return $(\mathsf{st}_{S,0}, \mathsf{st}_{R,0})$

$\mathsf{Send}(\mathsf{st}_S, m, f)$:
1 parse st_S as $(K, \mathsf{seqno}, \mathsf{buf})$
2 $\mathsf{buf} \leftarrow \mathsf{buf}||m$
3 $c \leftarrow \varepsilon$
4 while $|\mathsf{buf}| \geq \mathsf{il}$ do
5 $m' \leftarrow \mathsf{buf}[1, \ldots, \mathsf{il}]$
6 $\mathsf{buf} \leftarrow \mathsf{buf} \,\%\, m'$
7 $c' \leftarrow \mathsf{Enc}_K(\mathsf{seqno}, m')$
8 $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
9 $c \leftarrow c \,||\, |c'| \,||\, c'$ for $|c'| \in \{0,1\}^{\mathsf{ol}}$
10 if $f = 1$ and $\mathsf{buf} \neq \varepsilon$ then
11 $c' \leftarrow \mathsf{Enc}_K(\mathsf{seqno}, \mathsf{buf})$
12 $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
13 $c \leftarrow c \,||\, |c'| \,||\, c'$ for $|c'| \in \{0,1\}^{\mathsf{ol}}$
14 $\mathsf{buf} \leftarrow \varepsilon$
15 $\mathsf{st}_S \leftarrow (K, \mathsf{seqno}, \mathsf{buf})$
16 return (st_S, c)

$\mathsf{Recv}(\mathsf{st}_R, c)$:
1 parse st_R as $(K, \mathsf{seqno}, \mathsf{buf}, \mathsf{fail})$
2 if $\mathsf{fail} = 1$ then
3 return $(\mathsf{st}_R, \perp)$
4 $\mathsf{buf} \leftarrow \mathsf{buf}||c$
5 $m \leftarrow \varepsilon$
6 while $|\mathsf{buf}| \geq \mathsf{ol}$ do
7 parse $\mathsf{buf}[1, \ldots, \mathsf{ol}]$ as integer ℓ
8 if $|\mathsf{buf}| \geq \mathsf{ol} + \ell$ then
9 $len \leftarrow \mathsf{buf}[1, \ldots, \mathsf{ol}]$
10 $c' \leftarrow \mathsf{buf}[\mathsf{ol} + 1, \ldots, \mathsf{ol} + \ell]$
11 $\mathsf{buf} \leftarrow \mathsf{buf} \,\%\, len||c'$
12 $m' \leftarrow \mathsf{Dec}_K(\mathsf{seqno}, c')$
13 $\mathsf{seqno} \leftarrow \mathsf{seqno} + 1$
14 $m \leftarrow m||m'$
15 if $m' = \perp$ then
16 $\mathsf{fail} \leftarrow 1$
17 break
18 else
19 break
20 $\mathsf{st}_R \leftarrow (K, \mathsf{seqno}, \mathsf{buf}, \mathsf{fail})$
21 return (st_R, m)

Fig. 4: A 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.

not authenticated in TLS 1.3).[6] However, the TLS Record Protocol 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, though the exact nonce generation is left to be specified by the particular cipher suite in use.

The content type field in particular allows TLS to multiplex data streams for different purposes within a single connection stream, as TLS 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] That is, our approach of using a length field which is sent on the wire but not part of the authenticated associated data conforms with the approach adopted in TLS 1.3.

6 Conclusion

In this work we approached 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.

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 in [9] 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.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. Marc Fischlin is supported by the Heisenberg grant Fi 940/3-2 of the German Research Foundation (DFG). Kenneth Paterson is supported by EPSRC Leadership Fellowship EP/H005455/1 and by EPSRC grant EP/M013472/1. This work has been co-funded by the DFG as part of projects P2 and S4 within the CRC 1119 CROSSING and by the EU COST Action IC 1306.

References

1. 3rd Generation Partnership Project (3GPP): GSM, UMTS, and LTE standards, `http://www.3gpp.org`
2. Albrecht, M.R., Paterson, K.G., Watson, G.J.: Plaintext recovery attacks against SSH. In: 2009 IEEE Symposium on Security and Privacy. pp. 16–26. IEEE Computer Society Press (May 2009)
3. Badertscher, C., Matt, C., Maurer, U., Rogaway, P., Tackmann, B.: Augmented secure channels and the goal of the TLS 1.3 record layer. Cryptology ePrint Archive, Report 2015/394 (2015), `http://eprint.iacr.org/`

4. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: Online ciphers and the hash-CBC construction. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer (Aug 2001)
5. Bellare, M., Kohno, T., Namprempre, C.: 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)
6. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer (Dec 2000)
7. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy. pp. 98–113. IEEE Computer Society Press (May 2014)
8. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y.: Implementing TLS with verified cryptographic security. In: 2013 IEEE Symposium on Security and Privacy. pp. 445–459. IEEE Computer Society Press (May 2013)
9. Boldyreva, A., Degabriele, J.P., Paterson, K.G., Stam, M.: Security of symmetric encryption in the presence of ciphertext fragmentation. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 682–699. Springer (Apr 2012)
10. Boldyreva, A., Degabriele, J.P., Paterson, K.G., Stam, M.: On symmetric encryption with distinguishable decryption failures. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 367–390. Springer (Mar 2014)
11. Boldyreva, A., Taesombut, N.: Online encryption schemes: New security notions and constructions. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 1–14. Springer (Feb 2004)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), `http://eprint.iacr.org/2000/067`
13. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer (May 2001)
14. Degabriele, J.P., Paterson, K.G.: On the (in)security of IPsec in MAC-then-encrypt configurations. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 10. pp. 493–504. ACM Press (Oct 2010)
15. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (Aug 2008), `http://www.ietf.org/rfc/rfc5246.txt`, updated by RFCs 5746, 5878, 6176
16. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft (work in progress) (Jan 2015), `https://tools.ietf.org/id/draft-ietf-tls-tls13-04.txt`, Expires: July 7, 2015
17. Fielding, R., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230 (Proposed Standard) (Jun 2014), `http://www.ietf.org/rfc/rfc7230.txt`
18. Fouque, P.A., Joux, A., Martinet, G., Valette, F.: Authenticated on-line encryption. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 145–159. Springer (Aug 2004)
19. Institute of Electrical and Electronics Engineers, Inc.: IEEE Standard 801.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, `http://standards.ieee.org/about/get/802/802.11.html`

20. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer (Aug 2012)
21. Joux, A., Martinet, G., Valette, F.: Blockwise-adaptive attackers: Revisiting the (in)security of some provably secure encryption models: CBC, GEM, IACBC. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 17–30. Springer (Aug 2002)
22. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard) (Dec 2005), http://www.ietf.org/rfc/rfc4301.txt, updated by RFC 6040
23. Kohno, T., Palacio, A., Black, J.: Building secure cryptographic transforms, or how to encrypt and MAC. Cryptology ePrint Archive, Report 2003/177 (2003), http://eprint.iacr.org/2003/177
24. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer (Aug 2013)
25. Maurer, U., Tackmann, B.: On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 10. pp. 505–515. ACM Press (Oct 2010)
26. Namprempre, C.: Secure channels based on authenticated encryption schemes: A simple characterization. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 515–532. Springer (Dec 2002)
27. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the TLS record protocol. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 372–389. Springer (Dec 2011)
28. Paterson, K.G., Watson, G.J.: Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 345–361. Springer (May 2010)
29. Postel, J.: User Datagram Protocol. RFC 768 (INTERNET STANDARD) (Aug 1980), http://www.ietf.org/rfc/rfc768.txt
30. Postel, J.: Transmission Control Protocol. RFC 793 (INTERNET STANDARD) (Sep 1981), http://www.ietf.org/rfc/rfc793.txt, updated by RFCs 1122, 3168, 6093, 6528
31. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard) (Jan 2012), http://www.ietf.org/rfc/rfc6347.txt
32. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 02. pp. 98–107. ACM Press (Nov 2002)
33. Roskind, J.: QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/ (Dec 2013), retrieved on 2015-01-23
34. Shoup, V.: On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012 (1999), http://eprint.iacr.org/1999/012
35. Smyth, B., Pironti, A.: 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)
36. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard) (Jan 2006), http://www.ietf.org/rfc/rfc4251.txt
37. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard) (Jan 2006), http://www.ietf.org/rfc/rfc4253.txt, updated by RFC 6668