Robust Channels

Handling Unreliable Networks in the Record Layers of QUIC and DTLS

Marc Fischlin[1] Felix Günther[2] Christian Janson[1]

[1] Cryptoplexity, Technische Universität Darmstadt, Germany
[2] Department of Computer Science, ETH Zürich, Switzerland
marc.fischlin@cryptoplexity.de mail@felixguenther.info christian.janson@cryptoplexity.de

February 21, 2020

Abstract. The common approach in channel protocols is to rely on ciphertexts arriving in-order and to close the connection upon any rogue ciphertext. Cryptographic security models for channels generally reflect such design. This is reasonable when running atop lower-level transport protocols like TCP ensuring in-order delivery, as for example is the case with TLS or SSH. However, channels such as QUIC or DTLS which run over a non-reliable transport protocol like UDP, do not—and in fact cannot—close the connection if packages are lost or arrive in a different order. Those protocols instead have to carefully catch effects arising naturally in unreliable networks, usually by using a sliding-window technique where ciphertexts can be decrypted correctly as long as they are not misplaced too far.

To accommodate such handling of unreliable network messages, we introduce a generalized notion of *robustness* of cryptographic channels. This property can capture unreliable network behavior and guarantees that adversarial tampering cannot hinder ciphertexts that can be decrypted correctly from being accepted. We show that robustness is orthogonal to the common notion of integrity for channels, but together with integrity and chosen-plaintext security it provides a robust analogue of chosen-ciphertext security of channels. We then discuss two particularly interesting targets, namely the packet encryption in the record layer protocols of QUIC and of DTLS 1.3. We show that both protocols achieve the intended level of robust chosen-ciphertext security based on certain properties of their sliding-window techniques and on the underlying AEAD schemes.

Keywords. Secure channel · robustness · robust integrity · AEAD · QUIC · DTLS 1.3 · UDP

Contents

1 Introduction

Cryptographic channel protocols should provide confidentiality and authenticity in the presence of network adversaries. Consider for example the latest version of TLS in version 1.3 [21]. Ignoring subtle issues like fragmentation, the record layer protocol should ensure that the sender's ciphertexts $c_1, c_2, c_3, \ldots$ are correctly decrypted to the encapsulated messages at the receiver's side if they arrive in this order. Any (accidental or malicious) reordering or modifications of the ciphertexts should be detectable and, in case of a suspicious behavior, the standard specifies that the connection must be closed:

> `If the decryption fails, the receiver MUST terminate the connection with a "bad_record_mac" alert.`

TLS therefore assumes, or at least hopes, that packages are delivered reliably on the network. If a ciphertext accidentally gets lost on the transport layer then this most likely closes the channel connection on the application level. Put differently, this way of dealing with errors is closely associated to the TCP protocol as the underlying reliable, connection-oriented transport layer.

Other cryptographic channels like QUIC [13] or DTLS 1.3 [23], however, run atop an unreliable, datagram-oriented transport layer, UDP in these cases. From the channel's point of view this means that ciphertexts (or, fragments of ciphertexts) may be lost on the network or arrive in different order. Such protocols thus need to support more ample error handling. Usually, these protocols use a sliding-window technique to decrypt ciphertexts within the window, moving the window forward whenever a valid ciphertext beyond the current window arrives.

The sliding-window technique is interesting for the cryptographic channel for two reasons. One is that, currently, most cryptographic models for secure channels focus on the aborting type of protocols and thus do not touch upon the window technique (this includes, e.g., the initial formalization of stateful authenticated encryption [3, 4] used to analyze the SSH protocol [28], length-hiding authenticated encryption variants used to study the TLS protocol [19, 15], as well as more specialized models covering fragmentation [7], streaming [11], bidirectionality [18], or ratcheting security [14]). Another interesting aspect is that such protocols necessitate another property besides correctness and the common security notions which so far was mostly neglected.

1.1 Robustness of Channels as a First-class Property

In this work, we bring out *robustness* as a core property of cryptographic channels, which primarily focuses on protocols over an unreliable network, but also extends to reliable networks under active attacks. Robustness roughly says that malicious ciphertexts on the network cannot disturb the expected behavior of the channel. As a concrete example, robustness guarantees that an adversarial ciphertext injected in the channel cannot make the window of the sliding technique shift further, such that previous ciphertexts, which would otherwise have been inside the admissible window, would now get rejected.

We remark that robustness as a notion has so far not been captured by previous security definitions for channels when it comes to where it is most relevant, namely, for unreliable network transmission. In the realm of ratcheting [6], Jaeger and Stepanovs [14] discuss a restricted form of robustness for bidirectional channels as part of their correctness definition, but intentionally only treat reliable transport protocols. Boyd et al. [8], in their generalization of different levels of authentication/AEAD in a hierarchy similar to that introduced by Kohno, Palacio, and Black [16], come closest to the idea of a more fine-grained approach to different properties like reordering or dropping of ciphertexts. Likewise, Rogaway and Zhang [25] capture different level sets for permissible ordering for stateful authenticated encryption, capturing a hierarchy similar to [8, 16]. Yet, it turns out that QUIC [13, 27] and DTLS 1.3 [23] for example would be declared as insecure according to their models, for technically subtle reasons related to the way the sliding-window

technique can lead to previously rejected ciphertexts being later (rightfully) accepted when packet numbers are only partially transmitted.[1]

In a different light, Chen et al. [9] (and similarly Lychev et al. [17] in prior work for an early version) study the QUIC record layer as part of an overall ACCE-type analysis [15]. While their formalism treats QUIC as having no reordering and replay protection (level 1 in the hierarchy of [8]), they argue that sequence number authentication in QUIC "essentially" achieves TLS-like authentication and reordering protection. Our work aims at providing a more fine-grained analysis of the properties that sliding-window cryptographic channel protocols achieve over an unreliable network.

1.2 Defining General Robustness

Defining robustness as a general notion is delicate because we need to compare the behavior in presence of an active adversary to the expected behavior of the channel under non-malicious alternation due to the network, be it reliable or unreliable. To capture different expected channel behaviors like the ability to recover from ciphertext losses or from ciphertext reordering in a single definition, we parameterize the channel protocol by a predicate supp describing supported ciphertexts, i.e., ciphertexts which should be processed correctly by the channel. This predicate operates on the sequences of sent and received ciphertexts so far, and thus represents a global view on the network communication.

We show how the predicate supp allows to capture various scenarios for desired channel behavior, spanning both over reliable and unreliable networks. On the extreme ends this includes a strict ordering of ciphertexts at the receiver's side, as in TLS 1.3 over reliable networks, and (almost) no guarantees as in DTLS 1.2 with no replay protection. Our notion also allows to portray the different sliding-window techniques with both static or dynamic window sizes.

Introducing supp as a parameter initially also affects the correctness definition of a channel. Correctness then says that the protocol acts as expected on *supported* ciphertext sequences, and is now defined as a game between a weak network adversary which can only tamper with the order of ciphertexts. Once we have the advanced notion of correctness we can define robustness in a generalized way. Our notion, denoted ROB, compares the real behavior of the channel with the correct behavior that would be obtained when filtering out any maliciously modified or injected ciphertext by an active adversary. For a robust channel we expect both behaviors to be quasi identical, implying that the malicious ciphertexts cannot make the protocol deviate. In particular, if a channel uses sliding windows to identify admissible ciphertexts, then malicious network data cannot falsely modify the window boundaries.

1.3 Relationships of Security Notions

We then relate the notion of robustness to the classical notions of channel integrity and indistinguishability (under network-passive ($\mathsf{IND\text{-}CPA}$) and -active attacks ($\mathsf{IND\text{-}CCA}$)). For this we first lift the (stateful) notion of ciphertext integrity $\mathsf{INT\text{-}sfCTXT}$ [4] to our framework with the predicate supp, yielding our definition of INT. For chosen-ciphertext security we adopt the (stateless) $\mathsf{IND\text{-}CCA3}$ notion of Shrimpton [26] which combines integrity and confidentiality into a single game. The notion is called $\mathsf{INT\text{-}IND\text{-}CCA}$ in our setting.

We first argue that robustness and integrity are orthogonal in the sense that neither one implies the other. But we can define a combined notion called *robust integrity* ($\mathsf{ROB\text{-}INT}$) which is implied by

[1] More precisely, for basic authentication with replay protection both the model by Boyd et al. [8] ("level 2") as well as that by Rogaway and Zhang [25] ("basic level-set 1") demand that a scheme must reject any ciphertext that has already been processed earlier (to prevent replays). A scheme with a sliding-window technique may however first reject a ciphertext which is "too new" (too far ahead of the current window), but then later rightfully accept this ciphertext (when it is within the window) without opening up to replay attacks. Accepting the ciphertext the second time however violates the notions in [8, 25].

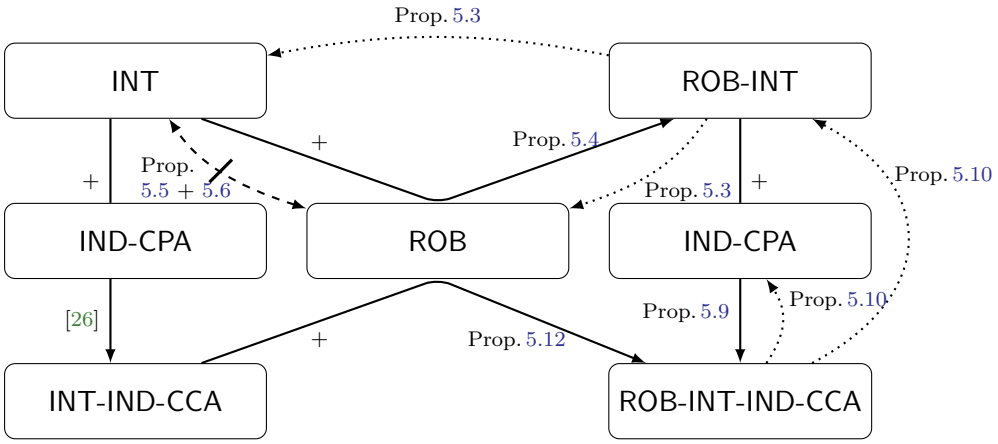

Figure 1: Overview over relationships of robustness, integrity, and indistinguishability notions for any fixed predicate supp. Solid arrows from A to C via B indicate the implication $A \wedge B \Rightarrow C$. Dotted arrows from A to B indicate explicitly shown implications $A \Rightarrow B$; further implications follow by transitivity. Dashed, struck-through arrows between A and B indicate separations of A and B.

both notions together, and vice versa implies both notions. We next define a notion ROB-INT-IND-CCA which is the "robust analogue" of INT-IND-CCA security for channels. We show that this robust notion can be achieved either by considering an IND-CPA secure channel which also provides robust integrity. Alternatively, one can add robustness to a INT-IND-CCA channel to derive the notion too. Conversely, ROB-INT-IND-CCA implies robust integrity and IND-CPA security and thus also INT-IND-CCA.

Our results about the relationship of the notions are summarized in Figure 1.

1.4 Robustness of QUIC and DTLS 1.3

Turning to the record layer protocols of QUIC and DTLS 1.3 we provide for both an abstract representation of their packet encryption as a cryptographic channel. Both protocols rely on an AEAD scheme to protect the payload. With minor differences both use sequence numbers as nonces for AEAD encryption but only transmit parts of the sequence number with the ciphertext. This of course requires the receiver to be able to recover the correct sequence number from the fraction transmitted with the ciphertext. This is accomplished by using a sliding window for determining the nearest sequence number to the received partial information. Remarkably, the window size is variable, e.g., can be between 1 and 4 bytes for QUIC, and chosen by the sender when creating the ciphertext. Note that this approach is different from previous approaches such as DTLS 1.2 which transmits the full nonce in clear.

The above window is required to determine the full packet number but does not necessarily provide protection against replay attacks. For instance, sending the same ciphertext twice immediately would yield the correct sequence number in both cases, since the window has not progressed too far the second time. Therefore, both protocols use another (fixed-size) sliding window on the receiver side to detect replayed ciphertexts. Both these replay-check windows reach backwards from the last valid sequence number on the receiver's side.

We go on to establish that QUIC achieves the intended level of robustness with respect to its supported in-window reordering with replay protection. Robustness of QUIC, beyond the appropriate encoding of (truncated) nonces within the sliding window, relies on the underlying AEAD scheme's integrity. Our proof actually shows robustness and integrity in one go, so that we can immediately deduce that the channel achieves the ROB-INT property above. Arguing that QUIC is IND-CPA is straightforward using

the confidentiality of the AEAD scheme, such that we can immediately conclude with our general results that the protocol provides robust combined integrity and indistinguishability (ROB-INT-IND-CCA). We achieve similar results in our robustness analysis of DTLS 1.3.

The robustness results for QUIC and DTLS 1.3 display some interesting behavior with respect to the security bounds. Namely, the fact that channels over unreliable networks need to keep open the connection when receiving (possibly maliciously) disordered ciphertexts gives an adversary multiple forgery attempts. This induces a non-tight security bound for robustness when reducing the security of AEAD scheme. This somehow matches experience with practical attacks that unreliable-network channels (like DTLS or QUIC) are easier to attack than those over reliable channels (like TLS), e.g., as observed in the Lucky Thirteen attack on the (D)TLS record protocols [1].

2 Preliminaries

We introduce some notation used throughout the paper. Additionally, we provide a brief recap of syntax and security of authenticated encryption with associated data [24].

2.1 Notation

We write a bit as $b \in \{0,1\}$ and a (bit) string as $s \in \{0,1\}^*$ with $|s|$ indicating its (binary) length. We implicitly interpret natural numbers as bit strings (of appropriate length) and vice versa, depending on the context, en-/decoding to/from big-endian bit binary encoding. For a bit string s and $i, j \in [1, |s|]$, we denote with $s[i]$ the i-th bit of s and with $s[i..j]$ the substring of s starting with the i-th bit and ending with, and including, the j-th bit, where for $j < i$ we set $s[i..j]$ to be the empty string, denoted by ε. We write $s \preccurlyeq t$ if s is a prefix of t, i.e., $t[1..|s|] = s$. By $s\|t$ we denote concatenation of two bit strings s, t as well as vector concatenation of two vectors s, t. For a vector s, we let the shorthand $s \Leftarrow x$ denote $s \leftarrow s\|(x)$, i.e., appending x as the next entry to s. For a bit string s of length $|s| = n$ and $m \in \mathbb{N} \cup \{0\}$ we denote by $s \ll m$ the string $s[1+m..n+m]\|0^{\min(m,n)}$ resulting from shifting in m zeros from the right. Note that the notation also covers the case that $m > n$ and hence the resulting (shifted) substring $s[1+m..n+m]$ is outside of the original range of the string. Hence this substring is initially empty and we concatenate a zero-string of length $\min(m,n)$ to assign each position in $s[1+m..n+m]$ a bit 0.

For a vector $t = (t_1, \ldots, t_n)$ we let $|t| = n$ denote the number of elements and $t[i] = t_i$ be the i-th entry. Let $()$ be the empty vector of length 0. We lift some of the operations on strings to vectors in the obvious way. That is, for two vectors $t = (t_1, \ldots, t_m)$ and $t' = (t_1, \ldots, t'_n)$ we write $t\|t' = (t_1, \ldots, t_m, t'_1, \ldots, t'_n)$ for the vector where we append t' element-wise to t. We also write $t \preccurlyeq t'$ if there exists a vector s such that $t\|s = t$. For a vector t we also write $x \in t$ if there exists an index $i \in [1..|t|]$ such that $x = t[i]$. For an m-element vector of n-element vectors $t = ((t_1^1, t_2^1, \ldots, t_n^1), \ldots, (t_1^m, t_2^m, \ldots, t_n^m))$ and $i \in [1,n]$ we denote by $t\langle i\rangle = (t_i^1, \ldots, t_i^m)$ the m-element vector consisting of all i-th entries of t's subvectors. Let t be a vector and $x = t[i]$ be a unique value in the vector, then we define $i = \mathsf{index}(x, t)$ to be index of x in t.

We provide all security results in terms of concrete security but occasionally also need asymptotic behaviors, e.g., when defining a general property like robustness ROB. In this case it is understood that all algorithms, including the adversary, then receive the security parameter in unary. In this case terms like "negligible" and "polynomial time" then refer to this security parameter.

2.2 Authenticated Encryption with Associated Data

We briefly recap the notion of authenticated encryption with associated data (AEAD) as introduced by Rogaway [24].

Definition 2.1 *An* authenticated encryption scheme with associated data $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ *is a pair of efficient algorithms such that:*

- *The deterministic encryption algorithm* $\mathsf{Enc}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \mathcal{M} \to \{0,1\}^*$ *takes as input a secret key* K*, a nonce* N*, associated data* AD*, and a message* m *outputting a ciphertext* c*.*
- *The deterministic decryption algorithm* $\mathsf{Dec}\colon \mathcal{K} \times \mathcal{N} \times \mathcal{H} \times \{0,1\}^* \to \mathcal{M} \cup \{\perp\}$ *takes as input a secret key* K*, a nonce* N*, associated data* AD*, and a ciphertext* c *outputting either a message* $m \in \mathcal{M}$ *or a dedicated error symbol* $\perp$ *indicating that the ciphertext is invalid.*

Sets $\mathcal{K}, \mathcal{N}, \mathcal{H}$ *and* $\mathcal{M}$ *denote respectively the key space, the nonce space, the associated data (header) space, and the message space associated to the scheme.*

Definition 2.2 *We say that an authenticated encryption scheme with associated data* AEAD *is* correct *if for all* $K \in \mathcal{K}$*,* $N \in \mathcal{N}$*,* $AD \in \mathcal{H}$ *and* $m \in \mathcal{M}$*, it holds that*

$$\Pr[\mathsf{Dec}(K, N, AD, \mathsf{Enc}(K, N, AD, m)) = m] = 1.$$

We define the two usual security notions of confidentiality and authenticity of an AEAD scheme. We start with defining confidentiality for an AEAD scheme in terms of an adversary $\mathcal{A}$ providing an input (N, AD, m_0, m_1) having a "small" advantage in determining whether it has received an encryption of the left message m_0 or the right message m_1. This notion is called IND-CPA. More formally the advantage is defined as $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{A}} = \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{A}} \Rightarrow 1\right] - 1/2$.

We define authenticity for an AEAD scheme through an adversary $\mathcal{A}$ with access to an encryption oracle $\textsc{Enc}(K, \cdot, \cdot, \cdot)$. We say that the adversary *forges* (a ciphertext) if it outputs a triple (N, AD, c) such that the decryption of the ciphertext is valid (i.e., $\mathsf{Dec}(K, N, AD, c) \neq \perp$) and the ciphertext c has not been an output of a previous query to the encryption oracle with nonce N and associated data AD. We define the advantage of the adversary against authenticity as the probability of it successfully forging, i.e., more formally $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{A}} = \Pr\left[\mathsf{Expt}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{A}} \Rightarrow 1\right]$.

3 Channels

In this section we give an augmented definition of channel protocols which will allow us to capture channel behavior over unreliable networks. As usual, a channel consists of three algorithms, for initialization, sending messages on the sender side, and receiving messages on the receiver side. However, we introduce two definitional twists that will allow us to capture *different* and possibly *dynamic* channel behaviors (depending on the underlying network): First, we parameterize the definition of correctness to capture different levels of supported variations in the ciphertext sequence (caused by the underlying network). Second, we provide the sending algorithm with an additional, auxiliary information (beyond the message to be transmitted) which is generic, but can capture dynamic sending behavior (like encoding we will see in QUIC and DTLS 1.3) that affects correctness properties.

Definition 3.1 (Channel protocol) *A* channel (protocol) $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *with associated sending and receiving state space* $\mathcal{S}_S$ *resp.* $\mathcal{S}_R$*, message space* $\mathcal{M} \subseteq \{0,1\}^{\leq M}$ *for some maximum message length* $M \in \mathbb{N}$*, auxiliary information space* $\mathcal{X}$*, error symbol* $\perp$ *with* $\perp \notin \{0,1\}^*$*, consists of three efficient algorithms defined as follows.*

- $\mathsf{Init}() \xrightarrow{\$} (\mathsf{st}_S, \mathsf{st}_R)$*. This probabilistic algorithm outputs initial sending and receiving states* $\mathsf{st}_S \in \mathcal{S}_S$*, resp.* $\mathsf{st}_R \in \mathcal{S}_R$*.*

- $\mathsf{Send}(\mathsf{st}_S, m, \mathsf{aux}) \xrightarrow{\$} (\mathsf{st}_S, c)$. *On input a sending state* $\mathsf{st}_S \in \mathcal{S}_S$*, a message* $m \in \mathcal{M}$*, and auxiliary information* $\mathsf{aux} \in \mathcal{X}$*, this (possibly) probabilistic algorithm outputs an updated state* $\mathsf{st}_S \in \mathcal{S}_S$ *and a ciphertext (or error symbol)* $c \in \{0,1\}^* \cup \{\perp\}$.

- $\mathsf{Recv}(\mathsf{st}_R, c) \to (\mathsf{st}_R, m)$. *On input a receiving state* $\mathsf{st}_R \in \mathcal{S}_R$ *and a ciphertext* $c \in \{0,1\}^*$*, this deterministic algorithm outputs an updated state* $\mathsf{st}_R \in \mathcal{S}_R$ *and a message (or error symbol)* $m \in \mathcal{M} \cup \{\perp\}$.

3.1 Correctness

We define correctness of a channel protocol in terms of a correctness experiment. In order to capture the underlying network possibly arbitrarily dropping or reordering (yet not modifying) packets, we define correctness with a "semi-malignant" adversary which determines the message inputs to the sender and the arrival order of ciphertexts (but cannot modify or inject ciphertexts). In the experiment we specify correctness with respect to a supported sequence of received ciphertexts, formalized through a predicate supp.[2] The predicate receives as input the (combined) sequence of sent ciphertexts and corresponding auxiliary information AC_S, the so far supportedly received ciphertexts C_R, as well as the next ciphertext c and decides whether this next ciphertext is supported by outputting true or false.

Some of the supported predicates we consider in the following implicitly assume that sent ciphertexts are unique. This is particularly the case for channels over unreliable networks that perform sliding-window processing and provide replay protection. This means that we limit our attention to channels with unique ciphertexts, which is accordingly encoded as part of a (thereby slightly more demanding) correctness definition. Note that when one wishes to treat the case of non-unique ciphertexts for particular support classes, such as those for QUIC and DTLS 1.3, then one needs to add a replay counting argument and requires additional information to be processed by then stateful support predicates, increasing the complexity of the model significantly. From a real-world channel perspective, however, ciphertext collision are extremely unlikely, in the order of the birthday bound for the (pseudo)random ciphertexts. Taking both observations into account, we think that it is reasonable to focus on unique ciphertexts as a reasonable trade-off between model complexity and the applicability to real-world channels. Furthermore, with this assumption we follow suit and are in line with prior works [8, 25]. We leave extending our framework of support classes towards capturing channels with non-unique ciphertexts as an avenue for future work.

The correctness experiment $\mathsf{Expt}^{\mathsf{correct(supp)}}_{\mathsf{Ch},\mathcal{A}}$ is provided in Figure 2 with a SEND and RECV oracle. At the beginning of the experiment, we run the initialization algorithm which outputs an initial sending and receiving state. Next, we initialize three empty lists AC_S, C_R and T for keeping track of processed data. Note that we record in AC_S the sequence of sent ciphertexts and the corresponding auxiliary information. In case we wish to refer to them separately, we do this via $A_S = AC_S\langle 1\rangle$ and $C_S = AC_S\langle 2\rangle$, respectively. We initially set the flag win to false which shall indicate successful attacks. Then the adversary is run with access to both oracles.

The oracle SEND on input a message and auxiliary input runs the Send algorithm to obtain a ciphertext and an updated sending state. Next the oracle checks whether the resulting ciphertext has been generated before by simply checking if it is stored in $AC_S\langle 2\rangle$. If this is the case then the flag win is set to true and the adversary wins. In case the algorithm outputs a new ciphertext then the oracle stores the generated ciphertext with the auxiliary input in the list AC_S, and in the list T we store the message-ciphertext pair. Finally, the oracle returns the ciphertext to the adversary.

The RECV oracle is queried upon an index j. In case the index is outside of the range of available ciphertexts stored in the list T, the oracle rejects this malformed request which is indicated by a dedicated

[2] Capturing correctness as a predicate-based experiment borrows from a similar approach taken by Backendal [2], combining the level-set concepts from [25] with channel correctness games as in [18, 14].

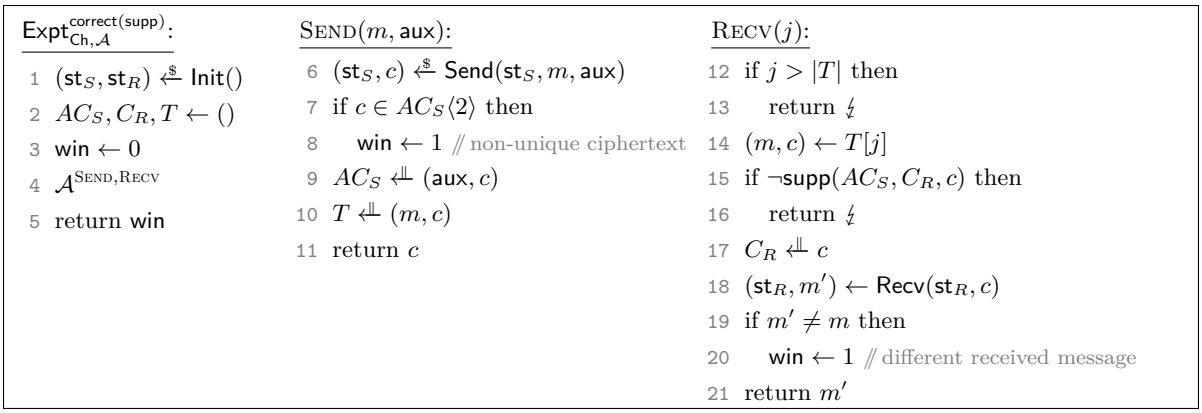

Figure 2: Experiment for correctness wrt. support class supp of a channel protocol Ch.

rejection symbol $\lightning \notin \{0,1\}^* \cup \{\bot\}$ and the oracle returns nothing.[3] If the index is valid, the oracle retrieves the corresponding message-ciphertext pair from T and then verifies whether the retrieved ciphertext is supported. If so, then C_R is accordingly updated and the oracle runs the Recv algorithm to obtain a message m' and an updated receiving state. The adversary succeeded to break correctness, captured by the flag win set to true, if the message m' returned from the receiving algorithm is different than the message m stored in the list T.

Definition 3.2 (Correctness of channels) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate, and experiment* $\mathsf{Expt}^{\mathsf{correct(supp)}}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 2.*

We define the advantage of $\mathcal{A}$ *in breaking* correctness wrt. supp *of* Ch *as*

$$\mathsf{Adv}^{\mathsf{correct(supp)}}_{\mathsf{Ch},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{correct(supp)}}_{\mathsf{Ch},\mathcal{A}} \Rightarrow 1\right],$$

and say that Ch *is (perfectly) correct wrt.* supp *if* $\mathsf{Adv}^{\mathsf{correct(supp)}}_{\mathsf{Ch},\mathcal{A}} = 0$ *for any* $\mathcal{A}$.

Note that one can easily define (ϵ-)correctness of the channel by requiring that the above advantage term is bounded by ϵ.

3.2 Examples of Support Classes

In the following, we discuss a few examples of different support classes which reflect different protocol purposes and environments (in terms of accepted reordering and replay protection). The examples illustrate the versatility of our supported predicate approach through a series of more and more complex designs; to assist understanding we underline changes wrt. to the previous predicate. Our examples in particular encompass the Internet security protocols TLS [21], DTLS [22, 23], and QUIC [13], but additionally include conceivable alternative support classes of channel protocols; some also link to authentication hierarchy levels put forward by Boyd et al. [8].

Recall that sometimes we separately refer to the sequence of sent ciphertexts and that of corresponding auxiliary information (both combined in AC_S) via $A_S = AC_S\langle 1\rangle$ and $C_S = AC_S\langle 2\rangle$, respectively. Furthermore with the notation $\mathsf{m} = \max\{\mathsf{index}(C_R[i], C_S) \mid i \in [1, |C_R|]\}$, we denote the highest index in C_S

[3] Observe that here, and in all following experiments, when returning $\lightning$, rejection happens purely as bookkeeping and is decided on information known to the adversary. As such, the dedicated symbol merely serves improved readability; returning $\bot$ would be equivalent.

among all supportedly received ciphertexts in C_R, and $\mathsf{n} = \mathsf{m} + 1$ denote the next "expected" ciphertext index on the receiver's end (one past m).

We require for any support predicate supp that $\mathsf{supp}(AC_S, C_R, c) = \texttt{false}$ for any $c \notin C_S$. This requirement encodes that supp is a correctness predicate and should only be true for genuinely sent ciphertexts.

No ordering. A channel that accepts packets in any order where the packets can also be duplicates; e.g., **DTLS 1.2 without replay protection** [22]. This corresponds to Level 1 in [8].

The corresponding predicate only ensures that each ciphertext was genuinely sent. Formally,

$$\mathsf{supp}_{no}(AC_S, C_R, c) := \Big[c \in C_S\Big].$$

No ordering with global anti-replay. A channel that accepts packets in any order, but rejects duplicates. This corresponds to Level 2 in [8].

The corresponding predicate ensures that each ciphertext was genuinely sent and not received before. Formally,

$$\mathsf{supp}_{no\text{-}r}(AC_S, C_R, c) := \Big[c \in C_S \wedge \underline{c \notin C_R}\Big].$$

While Boyd et al. [8] classify DTLS 1.2 with replay protection in their Level 2, corresponding to $\mathsf{supp}_{no\text{-}r}$, DTLS indeed suggests a sliding anti-replay window [22, Section 4.1.2.6] and hence cannot provide global (anti-)replay decisions. Indeed, DTLS would not achieve correctness wrt. $\mathsf{supp}_{no\text{-}r}$ since it rejects old ciphertexts past its replay window which $\mathsf{supp}_{no\text{-}r}$ would require to be supported. We hence consider a more fine-grained approach towards replay protection next.

No ordering with anti-replay window. A channel that accepts packets in a window of size w_r before m, or newer, rejecting duplicates; e.g., **DTLS 1.2 with replay protection** [22]. Here, w_r defines the anti-replay window (size) in which the channel checks for duplicates; any ciphertext older than what fits in this sliding window is conservatively rejected.

The corresponding predicate ensures that each ciphertext was genuinely sent, not received before, and is not older than w_r positions before the highest supportedly received ciphertext. Formally,

$$\mathsf{supp}_{no\text{-}r[w_r]}(AC_S, C_R, c) := \Big[c \in C_S \wedge c \notin C_R \wedge \underline{\mathsf{index}(c, C_S) \geq \mathsf{m} - w_r}\Big].$$

Observe that an infinite anti-replay window equals global anti-replay, i.e., $\mathsf{supp}_{no\text{-}r[\infty]} = \mathsf{supp}_{no\text{-}r}$.

Static sliding window. A channel that accepts packets in any order within a sliding window around the next expected ciphertext index n, reaching back w_b positions and forward w_f positions. Formally,

$$\mathsf{supp}_{sw[w_b,w_f]}(AC_S, C_R, c) := \Big[c \in C_S \wedge \underline{\mathsf{index}(c, C_S) \in [\mathsf{n} - w_b, \mathsf{n} + w_f]}\Big].$$

Observe that an infinite static window equals no ordering and that a zero-sized static window equals strict ordering, i.e., $\mathsf{supp}_{sw[\infty,\infty]} = \mathsf{supp}_{no}$ and $\mathsf{supp}_{sw[0,0]} = \mathsf{supp}_{so}$.

Static sliding window with anti-replay window. A channel that accepts packets in any order within a sliding window (reaching w_b positions backward and w_f positions forward) around the next expected ciphertext index, if they additionally check as non-duplicates within an anti-replay window of size w_r.

The corresponding predicate combines w_r and w_b in its in-window check since the received ciphertext index must be greater than or equal to both $\mathsf{n} - w_b$ and $\mathsf{m} - w_r = \mathsf{n} - (w_r + 1)$. Formally,

$$\mathsf{supp}_{sw[w_b,w_f]\text{-}r[w_r]}(AC_S, C_R, c) := \Big[c \in C_S \wedge c \notin C_R \wedge \mathsf{index}(c, C_S) \in [\mathsf{n} - \min(w_b, w_r + 1), \mathsf{n} + w_f]\Big].$$

Observe that an infinite static window equals no ordering with the same anti-replay window, i.e., $\mathsf{supp}_{sw[\infty,\infty]\text{-}r[w_r]} = \mathsf{supp}_{no\text{-}r[w_r]}$.

Dynamic sliding window with anti-replay window. A channel that accepts packets in any order within a sliding window (around the expected next ciphertext index n) that is dynamically determined for each sent ciphertext, if they additionally check as non-duplicates within an anti-replay window of size w_r; e.g., **DTLS 1.3 with replay protection** [23] and **QUIC** [13].

We assume the dynamic backward and forward window size w_b, resp. w_f, are encoded in the auxiliary information provided to Send as tuple $\mathsf{aux} = (w_b, w_f) \in \mathcal{X}$. (For concrete instances see the treatments of QUIC and DTLS 1.3 in Section 6 and Section 7, respectively.) The supported predicate then individually determines for each ciphertext c whether it was received within the dynamic window determined by w_b^c, w_f^c specified upon sending c. Again, the backward window combines w_b^c and the anti-replay window size w_r. Formally,

$$\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) := \Big[c \in C_S \wedge c \notin C_R \wedge \mathsf{index}(c, C_S) \in [\mathsf{n} - \min(w_b^c, w_r + 1), \mathsf{n} + w_f^c]\Big],$$

where $(w_b^c, w_f^c) = A_S[\mathsf{index}(c, C_S)]$.

Observe that for a single-entry auxiliary information space $\mathcal{X} = \{(w_b, w_f)\}$, dynamic and static sliding window (with same replay window) coincide, i.e., $\mathsf{supp}_{dw\text{-}r[w_r]} = \mathsf{supp}_{sw[w_b,w_f]\text{-}r[w_r]}$.

As discussed in the previous section, when considering non-unique ciphertexts then the above support classes get significantly more complicated. For example, let us consider the support predicate that accepts packets in any order but rejects duplicates, namely $\mathsf{supp}_{no\text{-}r}(AC_S, C_R, c)$, which only outputs 1 if the ciphertext to be checked is contained in C_S but *not* in C_R. If we consider non-unique ciphertexts then any repeated ciphertext (in C_S) would be handled as a replay and hence the channel would have to discard such repeated ciphertext on the receiver's side even though it was indeed a genuine output repeated by the sender. To avoid requiring such rejection we would need to add a counting argument into the support predicate, i.e., we would need to require that the sequence C_S contains at least one more non-unique ciphertext than the sequence C_R, such that when a repeated ciphertext arrives now the channel should accept it.

While such change can somewhat easily be introduced for this simplest unordered support predicate $\mathsf{supp}_{no\text{-}r}$, matters get much more complicated for more complex support predicates: In particular for those capturing (dynamic) sliding windows which we are primarily interested in for our later analyses of QUIC and DTLS 1.3, capturing non-unique ciphertext would require introducing state to the support predicate. For the sake of keeping the complexity of support predicates somewhat manageable, we therefore restrict our focus to channels with unique ciphertexts (taking according arguments into account when analyzing QUIC and DTLS 1.3).

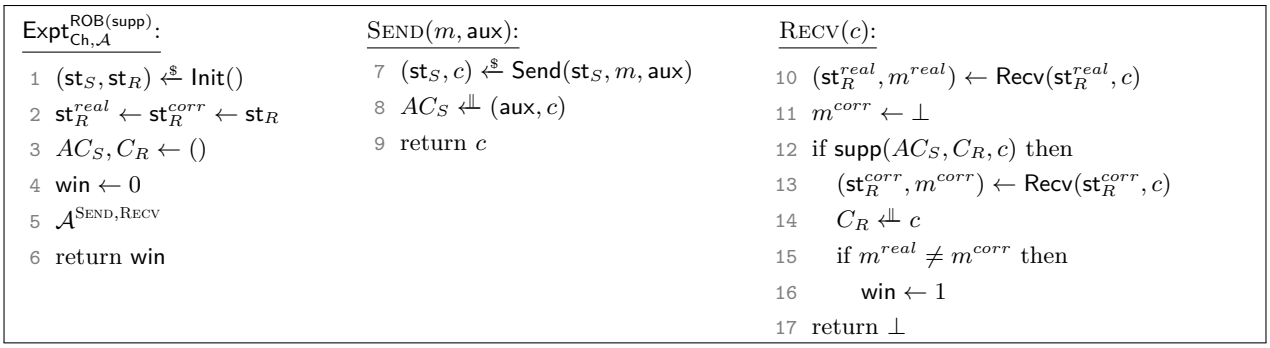

Figure 3: Experiment for robustness wrt. support class supp of a channel protocol Ch.

4 Robust Channels

We now introduce our new notion of robustness for channel protocols. With this notion, we aim to model behavior that is already present in protocols like QUIC [13] and DTLS 1.3 [23], namely that ciphertexts can be delivered out-of-order within a certain (sliding) window, and in addition the receiver is robust against any interleaved ciphertext which do not fit into the window (or are even maliciously crafted by a network adversary). Robustness here refers to a channel's property to filter out any misplaced ciphertexts and correctly receive those ciphertexts that fit into the supported order.

We define robustness according to Figure 3. The experiment processes the received sequence of ciphertexts (into which the adversary is free to inject forged ciphertexts) through two separate receiving instances: The first, "real" receiving instance process is called on every received ciphertext (Line 10). The second, "correct" receiving instance is only given those ciphertexts that are supported according to the predicate supp (Lines 12 and 13). Robustness then demands that, on any supported ciphertext, the output of the "correct" receiving instance never differs from the "real" instance's output.

Definition 4.1 (Robustness of channels, ROB) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate, and experiment* $\mathsf{Expt}^{\mathsf{ROB(supp)}}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 3.*

We define the advantage of $\mathcal{A}$ *in breaking* robustness wrt. supp *of* Ch *as*

$$\mathsf{Adv}^{\mathsf{ROB(supp)}}_{\mathsf{Ch},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{ROB(supp)}}_{\mathsf{Ch},\mathcal{A}} \Rightarrow 1\right],$$

and say that Ch *is robust wrt.* supp *if* $\mathsf{Adv}^{\mathsf{ROB(supp)}}_{\mathsf{Ch},\mathcal{A}}$ *is negligible for any polynomial-time* $\mathcal{A}$*.*

5 Robustness, Integrity, and Indistinguishability

In this section we relate the notion of robustness to the classical notions of channel integrity and indistinguishability.

5.1 Defining Robustness and Integrity

Robustness of a channel allows to make a statement about the behavior of the channel on supported sequences, even if there are malicious ciphertexts in-between. We can also define a notion of integrity of channels over unreliable networks. This notion says that the receiver should *not* decrypt any ciphertext to a valid message, unless the ciphertext is supported. We first give a "classical" definition of integrity and then introduce an equivalent version which is cast in the style of our notion of robustness.

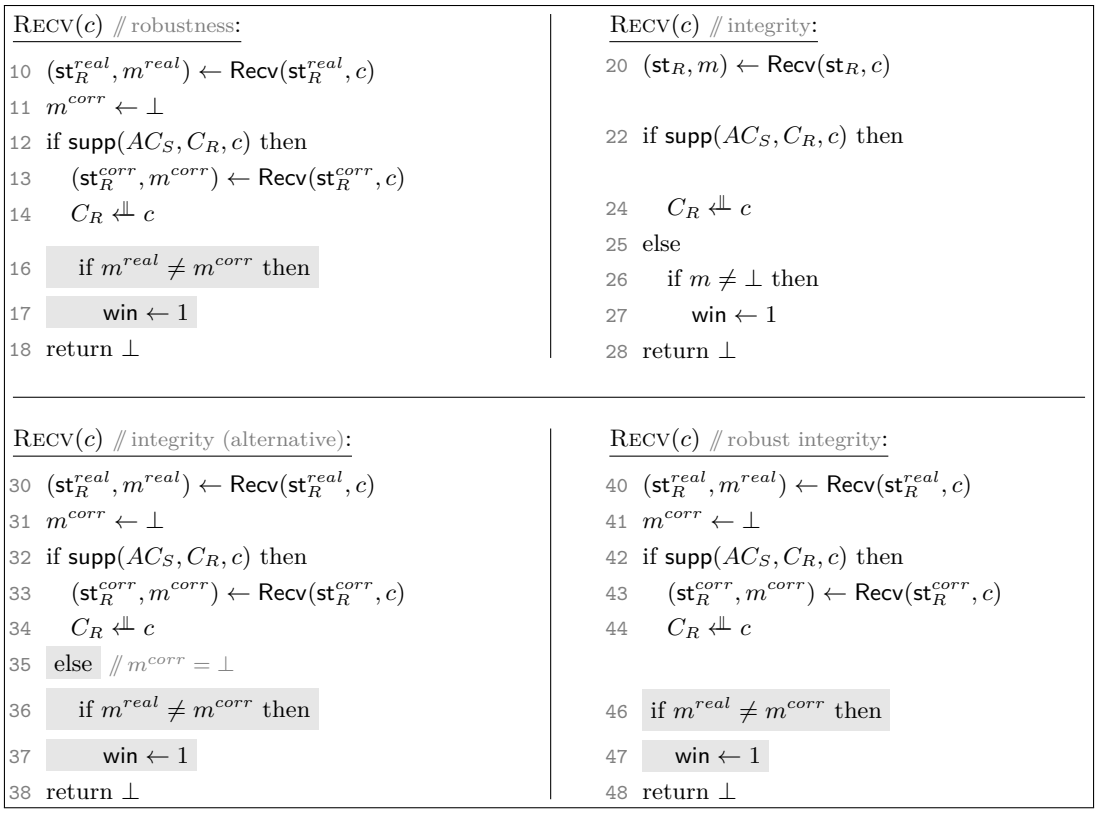

Figure 4: Receiver oracles in the experiments for robustness (upper left), integrity (upper right), alternative integrity (lower left) and robust integrity (lower right) wrt. support class supp of a channel protocol Ch. Differences are highlighted in grey boxes.

On the upper right-hand side of Figure 4, we present the notion of integrity, and in the lower left-hand side our alternative notion of integrity. Note that the given experiment $\mathsf{Expt}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}$ only differs in the receive oracle compared to the robustness experiment (cf. Figure 3) and hence we simply provide the details of the receive oracle as a description of the experiment. In more detail, in this experiment we check only on unsupported ciphertexts if they decrypt to a valid message m^{real} different from m^{corr}. The latter is always set to $\bot$ in Line 31 and not changed for unsupported ciphertexts, because the if-clause in Line 32 is skipped.

We first argue that the notions of integrity, the classical one and our alternative notion, are equivalent. This is easy to see since in both experiments the receiver's oracle behavior on supported ciphertexts is identical—in our notion one only performs the redundant step of decrypting—and on unsupported ciphertexts the receiver checks the decrypted message against $\bot$. Hence, we can define integrity with respect to either receive oracle:

Definition 5.1 (Integrity of channels, INT) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate, and experiment* $\mathsf{Expt}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as on the upper right hand side or lower left hand side in Figure 4.*

We define the advantage of $\mathcal{A}$ *in breaking* integrity wrt. supp *of* Ch *as*

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

We say that Ch *provides integrity wrt.* supp *if* $\mathsf{Adv}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}$ *is negligible for any polynomial-time* $\mathcal{A}$.

The lower right hand side of Figure 4 shows a combination of both notions which we call *robust integrity*. The difference compared to integrity is that we now check if the message decrypts to the expected value on *both* supported and unsupported ciphertexts.

Definition 5.2 (Robust integrity of channels, ROB-INT) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate, and experiment* $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as on the lower right hand side in Figure 4.*

We define the advantage of $\mathcal{A}$ *in breaking* robust integrity wrt. supp *of* Ch *as*

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

and say that Ch *achieves robust integrity wrt.* supp *if* $\mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}$ *is negligible for any polynomial-time adversary* $\mathcal{A}$*.*

5.2 Relating Robustness and Integrity

We next show that robustness and integrity imply robust integrity and vice versa. Robustness and integrity individually are incomparable, though. We start by showing that robust integrity implies the other two notions:

Proposition 5.3 $(\mathsf{ROB\text{-}INT} \Rightarrow \mathsf{ROB} \wedge \mathsf{INT})$ *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate. Then for any adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{ROB}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}, \quad \mathsf{Adv}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}.$$

Proof. The proposition is straightforward from the experiments. Consider an adversary against robustness resp. against integrity. Consider the first query c to the receive oracle which causes win to become true. Up to this point all three experiments for integrity, robustness, and robust integrity display an identical behavior, always returning $\perp$ in the receiver's oracle and keeping the same lists AC_S, C_R of sent ciphertexts and auxiliary information and (supportedly) received ciphertexts. If an adversary now triggers win to become 1 in either the robustness experiment (on a supported ciphertext) or the integrity experiment (on an unsupported ciphertext), then the if-clause in Line 46 of the robust-integrity experiment (cf. Figure 4) also sets win to 1. □

The converses do not necessarily hold. Assume that we have a channel which processes supported ciphertexts as expected, but on unsupported ciphertexts always outputs the message $m = 0$. This channel would be robust because it works correctly on supported ciphertexts, but it does not provide integrity nor robust integrity, because it returns the message $m = 0 \neq \perp$ on all unsupported ciphertexts. Note that this channel would nonetheless be correct.

Next, assume that we have a channel which, when receiving the first unsupported ciphertext will output $\perp$ but from then on decrypt all supported ciphertexts to message $m = 0$. This behavior is encoded in the channel's state. This channel is still correct because the bad event is never triggered on genuine ciphertext sequences. Furthermore, the channel provides integrity because on all unsupported ciphertexts the behavior correctly returns errors $\perp$. However, the channel clearly does not provide robustness nor robust integrity because of the wrong decryption on supported ciphertexts after the first unsupported ciphertext, returning $m = 0 \neq \perp$ on all such ciphertexts.

The above examples show that robustness or integrity alone do not suffice to guarantee robust integrity. In combination, though, they achieve the stronger notion as the next proposition shows.

Proposition 5.4 ($\mathsf{ROB} \wedge \mathsf{INT} \Rightarrow \mathsf{ROB\text{-}INT}$) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate. Then for any adversary* $\mathcal{A}$ *we have*

$$\mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} + \mathsf{Adv}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}.$$

Proof. Assume that we have an adversary $\mathcal{A}$ which causes win to become true because the if-clause $m^{real} \neq m^{corr}$ in Line 46 of Figure 4 is satisfied. Consider the first query where this happens. Up to this point all experiments behave identically. In particular, the sequence C_R is the same in all runs in all cases. This implies that the set of supported ciphertexts is also identical up till then. There are now two cases when the robust integrity adversary triggers the bad event:

- Either the call is for a supported ciphertext c, in which case we will run the "correct" receiver to get m^{corr} and will thus also reach Line 16 in the robustness experiment (cf. Figure 4) for the same value m^{corr}, setting win to true there.
- Or, the call is for an unsupported ciphertext c, in which case $m^{corr} = \bot$ and we will reach Line 36 in the integrity experiment (cf. Figure 4), and win will become true there.

Hence, any break in the robust integrity experiment means that the adversary breaks robustness or integrity, such that we can bound the advantage for the former by the sum of the advantages for the latter. □

We give a more formal separation of robustness and integrity here. We assume a support predicate supp which only relies on the received ciphertexts C_R and the given ciphertext c, e.g., if it merely checks that the sequence number in c exceeds the one in the last entry of C_R. In particular, we also assume that $\mathsf{supp}(c, c) = 0$ for any c. We assume for simplicity that the original channel is perfectly correct:

Proposition 5.5 ($\mathsf{ROB} \not\Rightarrow \mathsf{INT}$) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a perfectly correct channel,* supp *a correctness support predicate which only depends on* C_R *and* c *and where* $\mathsf{supp}(c, c) = 0$ *for any* c*. Then there is a channel protocol* $\mathsf{Ch}^* = (\mathsf{Init}^*, \mathsf{Send}^*, \mathsf{Recv}^*)$ *such that for any adversaries* $\mathcal{A}, \mathcal{B}$ *we have*

$$\mathsf{Adv}^{\mathsf{correct}(\mathsf{supp})}_{\mathsf{Ch}^*,\mathcal{A}} = 0, \quad \mathsf{Adv}^{\mathsf{ROB}(\mathsf{supp})}_{\mathsf{Ch}^*,\mathcal{B}} = 0,$$

but there is an adversary $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch}^*,\mathcal{C}} = 1.$$

Proof. The new channel Ch^* only modifies the receiver algorithm Recv from Ch and leaves Init and Send essentially unchanged, only the initial receiver state becomes $\mathsf{st}^*_R = (\mathsf{st}_R, ())$. Define

$\underline{\mathsf{Recv}^*(\mathsf{st}^*_R, c)\text{:}}$
1 parse $\mathsf{st}^*_R = (\mathsf{st}_R, C_R)$
2 if $\mathsf{supp}(C_R, c)$ then
3 $\quad(\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
4 $\quad C_R \Vdash c$
5 $\quad\mathsf{st}^*_R \leftarrow (\mathsf{st}_R, C_R)$
6 $\quad$return (st^*_R, m)
7 else
8 $\quad$return $(\mathsf{st}^*_R, 0)$

We first argue that correctness is preserved. This follows as the receiver in the correctness experiment is only invoked on supported ciphertexts and only then C_R is updated. This is identical to the behavior of Recv^*. The condition on the sender side is the same as before because our modified channel does not change Send.

For robustness note that both receiver states, $\mathsf{st}_R^{*,real}$ and $\mathsf{st}_R^{*,corr}$ are always in-sync when running the experiment for the modified channel, because unsupported ciphertexts do not change the receiver state. It follows that both states always yield the same answers.

Finally consider adversary $\mathcal{C}$ against integrity of Ch^*. This adversary sends an arbitrary ciphertext c *twice* to the receiver oracle. At least the second query will be unsupported by assumption about supp, such that Recv^* returns the message 0. The integrity game then checks that $m^{real} = 0 \neq m^{corr} = \perp$ and sets win to true. □

Proposition 5.6 ($\mathsf{INT} \not\Rightarrow \mathsf{ROB}$) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a perfectly correct channel,* supp *a correctness support predicate which only depends on* C_R *and* c *and where* $\mathsf{supp}(c, c) = 0$ *for any* c*. Then there is a channel protocol* $\mathsf{Ch}^* = (\mathsf{Init}^*, \mathsf{Send}^*, \mathsf{Recv}^*)$ *such that for any adversaries* $\mathcal{A}, \mathcal{B}$ *we have*

$$\mathsf{Adv}^{\mathsf{correct}(\mathsf{supp})}_{\mathsf{Ch}^*,\mathcal{A}} = 0, \quad \mathsf{Adv}^{\mathsf{INT}(\mathsf{supp})}_{\mathsf{Ch}^*,\mathcal{B}} = 0,$$

but there is an adversary $\mathcal{C}$ *such that*

$$\mathsf{Adv}^{\mathsf{ROB}(\mathsf{supp})}_{\mathsf{Ch}^*,\mathcal{C}} = 1.$$

Proof. The channel protocol Ch^* alters the receiver algorithm Recv from Ch and leaves Init and Send unmodified, only the initial receiver state becomes $\mathsf{st}_R^* = (\mathsf{st}_R, (), 0)$. Define

$\underline{\mathsf{Recv}^*(\mathsf{st}_R^*, c)}$:
1 parse $\mathsf{st}_R^* = (\mathsf{st}_R, C_R, d)$
2 if $\mathsf{supp}(C_R, c)$ then
3 $\quad C_R \Lleftarrow c$
4 $\quad$ if $d = 1$ then
5 $\quad\quad m \leftarrow 0$
6 $\quad$ else
7 $\quad\quad (\mathsf{st}_R, m) \leftarrow \mathsf{Recv}(\mathsf{st}_R, c)$
8 $\quad \mathsf{st}_R^* \leftarrow (\mathsf{st}_R, C_R, d)$
9 $\quad$ return (st_R^*, m)
10 else
11 $\quad$ return $((\mathsf{st}_R, C_R, 1), \perp)$

Correctness is preserved because the receiver in the correctness experiment is only executed on supported ciphertexts, such that the bit d remains 0 and the receiver algorithms answers faithfully for all queries. The sender is unchanged such that correctness violations in the sender oracle cannot occur here as well. Integrity holds unconditionally because on unsupported ciphertexts the receiver always returns the error symbol $\perp$, such that always $m^{real} = m^{corr} = \perp$ in the experiment in this case, leaving win to be false throughout the entire experiment.

The adversary $\mathcal{C}$ against robustness first calls the sender about $m = 1$ to get a ciphertext c. Then it calls the receiver oracle about this c twice. Since this ciphertext is unsupported in the second call, it turns the receiver's state $\mathsf{st}_R^{*,real}$ to $(\mathsf{st}_R, c, 1)$, but leaving $\mathsf{st}_R^{*,corr}$ unaltered from the previous valid call. Then the adversary calls the sender about the message $m = 1$ to get a ciphertext c' and forwards c' to the receiver oracle. According to correctness of the original channel the ciphertext c' must be supported and create the message $m^{corr} = 1$; the reason is that from the receiver's viewpoint with state st_R^{corr} it has

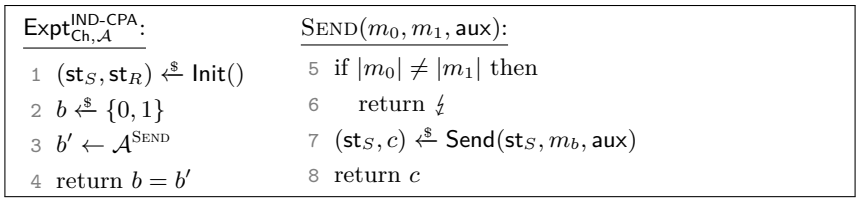

Figure 5: Experiment for IND-CPA of a channel protocol Ch.

received two genuine ciphertexts so far such that correctness ensures that the message decrypts correctly. Our modified receiver state $\mathsf{st}_R^{*,real}$, on the other hand, yields $m^{real} = 0$ by construction, because $d = 1$ at this point. Hence our adversary wins the robustness game with probability 1. □

5.3 Robustness and Chosen Ciphertext Security

Let us begin this section with defining IND-CPA security.

Definition 5.7 (IND-CPA) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel and experiment* $\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 5.*

We define the advantage of $\mathcal{A}$ *in breaking* indistinguishability of chosen plaintexts *of* Ch *as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2},$$

and say that Ch *is* IND-CPA*-secure if* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{A}} \approx 0$ *for any polynomial-time* $\mathcal{A}$*.*

We next define ROB-INT-IND-CCA for channels which follows the paradigm to combine confidentiality and integrity into a single experiment, called IND-CCA3 in [26]. The formal details are displayed in Figure 6. The idea is to return a message different from m by the receiver oracle if the adversary has broken robustness or integrity via the submitted ciphertext c, and if $b = 1$ (whereas we always return $\perp$ if $b = 0$). This enables the adversary to determine the bit b when breaking robust integrity. For this we overwrite m^{real} with $\perp$ if $m^{real} = m^{corr}$ and no break has occurred (Line 20). But if the messages are distinct we return the message which is not $\perp$ (Line 22).

Definition 5.8 (Robust integrity/indistinguishability of channels, ROB-INT-IND-CCA) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate, and experiment* $\mathsf{Expt}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 6.*

We define the advantage of $\mathcal{A}$ *in breaking* robust integrity/indistinguishability of chosen ciphertexts wrt. supp *of* Ch *as*

$$\mathsf{Adv}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2},$$

and say that Ch *is* ROB-INT-IND-CCA*-secure wrt.* supp *if* $\mathsf{Adv}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}}$ *is negligible for any polynomial-time adversary* $\mathcal{A}$*.*

The next proposition says that a channel achieves ROB-INT-IND-CCA if it has both robust integrity (ROB-INT) and IND-CPA.

Proposition 5.9 (ROB-INT $\wedge$ IND-CPA $\Rightarrow$ ROB-INT-IND-CCA) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate. Then for any adversary* $\mathcal{A}$ *there exist adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *with comparable run time such that*

$$\mathsf{Adv}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB\text{-}INT(supp)}}_{\mathsf{Ch},\mathcal{B}} + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{C}}.$$

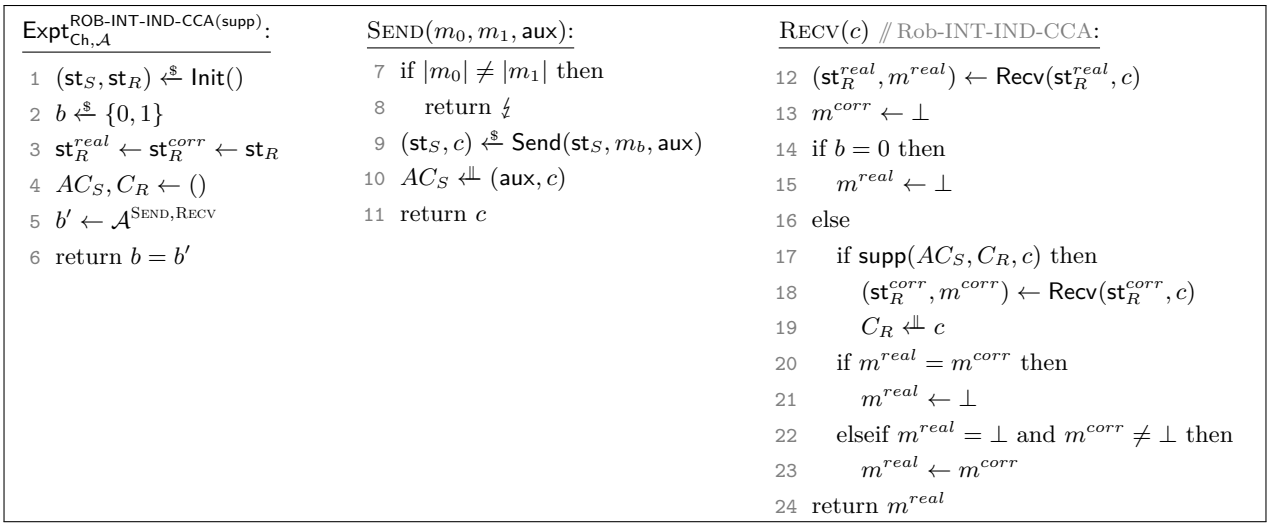

Figure 6: Experiment for ROB-INT-IND-CCA wrt. support class supp of a channel protocol Ch.

Proof. Consider an attacker $\mathcal{A}$ in experiment $\mathsf{Expt}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}}$ against the ROB-INT-IND-CCA property. Assume that we change $\mathcal{A}$'s experiment by letting the receiver oracle in the experiment always return $\bot$. We claim that the difference is negligible from $\mathcal{A}$'s perspective, since the oracle never returns a message $m \neq \bot$ with overwhelming probability. We argue this by embedding $\mathcal{A}$ into an adversary $\mathcal{B}$ playing the robust integrity experiment $\mathsf{Expt}^{\mathsf{ROB\text{-}INT(supp)}}_{\mathsf{Ch},\mathcal{B}}$. If the receiver oracle in $\mathcal{A}$'s original attack ever returns $m \neq \bot$ then we claim that $\mathcal{B}$ immediately breaks (robust) integrity.

Adversary $\mathcal{B}$ initially picks a bit $b \xleftarrow{\$} \{0,1\}$ and starts a simulation of $\mathcal{A}$. Any Send call (m_0, m_1, aux) of $\mathcal{A}$ is answered by first checking that $|m_0| = |m_1|$, returning $\lightning$ if not, and otherwise forwarding (m_b, aux) to $\mathcal{B}$'s own oracle Send, feeding the reply back to $\mathcal{A}$. Adversary $\mathcal{B}$ answers any query c of $\mathcal{A}$ to the receiver oracle as follows: If $b = 0$ then $\mathcal{B}$ immediately returns $\bot$. Else it sends c to its own oracle Recv and receives $\bot$. It returns $\bot$ to $\mathcal{A}$.

First observe that, up to the first query of $\mathcal{A}$ to Recv yielding a message $m \neq \bot$ as output, $\mathcal{B}$'s simulation perfectly mimics the actual attack from $\mathcal{A}$'s point of view in the sense that even the concrete executions match. In particular, the lists of sent and received ciphertexts are identical. Assume that $\mathcal{A}$ in its original attack at some point obtains a response distinct from $\bot$ from the (genuine or simulated) receiver oracle for a ciphertext c. This can only happen if $b = 1$ and

- the decrypted message m^{real} is different from $\bot$ and from m^{corr} (Line 20), or
- $m^{real} = \bot$ but $m^{corr} \neq \bot$ (Line 22).

In this case, the receiver's oracle of $\mathcal{B}$ will evaluate the condition $m^{real} \neq m^{corr}$ in Line 46 (cf. Figure 4) to true and make win become 1. It follows that $\mathcal{B}$ wins against robust integrity if $\mathcal{A}$ ever makes the receiver oracle return a message $m \neq \bot$.

Given that we have now turned the receiver oracle in $\mathcal{A}$'s attack into the always rejecting $\bot(\cdot)$ oracle, we can easily wrap $\mathcal{A}$ into an adversary $\mathcal{C}$ against the IND-CPA property. For this we let $\mathcal{C}$ answer each receiver query of $\mathcal{A}$ with $\bot$, and let $\mathcal{C}$ relay all send queries faithfully. It follows that $\mathcal{A}$'s advantage is bounded by $\mathcal{C}$'s advantage. □

In the following, we show that robust integrity (ROB-INT) and IND-CPA are both necessary to achieve the ROB-INT-IND-CCA property. Additionally we continue showing that if we establish robustness for an

INT-IND-CCA-secure channel then the channel also provides ROB-INT-IND-CCA. This gives an alternative construction and proof method for such channels.

Proposition 5.10 (ROB-INT-IND-CCA $\Rightarrow$ ROB-INT $\wedge$ IND-CPA) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate. Then for any adversary* $\mathcal{A}$ *there exists adversary* $\mathcal{B}$ *with comparable run time such that we have*

$$4 \cdot \mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{B}},$$
$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{B}}.$$

Proof. Clearly, if we can break IND-CPA security of the channel, then we also break ROB-INT-IND-CCA security (by omitting calls to the receiver oracle). We next argue that we can break ROB-INT-IND-CCA if we can break robust integrity, too. Assume that we have an attacker $\mathcal{A}$ against robust integrity. We build an attacker $\mathcal{B}$ against the ROB-INT-IND-CCA property. Algorithm $\mathcal{B}$ simulates $\mathcal{A}$ by answering each call (aux, m) to the SEND oracle by forwarding (aux, m, m) to its own SEND oracle and handing back the ciphertext c. Each of $\mathcal{A}$'s call to RECV is forwarded by $\mathcal{B}$ to its own receiver oracle, and $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. If the receiver oracle at some point returns a message $m \neq \perp$ to $\mathcal{B}$ then $\mathcal{B}$ immediately outputs 1; in any other case it outputs a random bit.

Note that $\mathcal{B}$ perfectly simulates the environment for $\mathcal{A}$'s attack, independently of the secret bit b. By assumption, $\mathcal{A}$ hence breaks robust integrity in the simulation with the same probability. Whenever this happens and $b = 1$ then $\mathcal{B}$ obtains a message $m \neq \perp$ and thus outputs $b' = 1$. If we denote this event, that $\mathcal{A}$ breaks integrity and that $b = 1$, by SUCC, then the probability of $\mathcal{B}$ predicting b correctly if lower bounded by the sum that the event happens plus the probability that the event does not occur but $\mathcal{B}$'s random guess is correct:

$$\begin{aligned}\Pr[b' = b] &\geq \Pr[\textsc{Succ}] + \frac{1}{2} \cdot \Pr[\overline{\textsc{Succ}}] \\ &= \frac{1}{2} + \frac{1}{2} \cdot \Pr[\textsc{Succ}] \\ &\geq \frac{1}{2} + \frac{1}{4} \cdot \mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}},\end{aligned}$$

where the latter follows since $\mathcal{A}$'s success probability is independent of the random bit b in $\mathcal{B}$'s experiment. □

We next show that instead of starting from IND-CPA and using robust integrity to achieve ROB-INT-IND-CCA, we can also add robustness to a channel which already provides INT-IND-CCA to arrive there. One option to show this would be to argue that INT-IND-CCA implies integrity. This would allow to conclude that robustness with integrity implies robust integrity, and that the latter yields ROB-INT-IND-CCA together with the IND-CPA security of the channel. Here, we show the security of the transform directly starting from INT-IND-CCA and adding robustness.

Definition 5.11 (INT-IND-CCA) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate, and experiment* $\mathsf{Expt}^{\mathsf{INT\text{-}IND\text{-}CCA}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}}$ *for an adversary* $\mathcal{A}$ *be defined as in Figure 7.*

We define the advantage of $\mathcal{A}$ *in breaking* integrity/indistinguishability of chosen ciphertexts wrt. supp *of* Ch *as*

$$\mathsf{Adv}^{\mathsf{INT\text{-}IND\text{-}CCA}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} := \Pr\left[\mathsf{Expt}^{\mathsf{INT\text{-}IND\text{-}CCA}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} \Rightarrow 1\right] - \frac{1}{2},$$

and say that Ch *is* INT-IND-CCA*-secure wrt.* supp *if* $\mathsf{Adv}^{\mathsf{INT\text{-}IND\text{-}CCA}(\mathsf{supp})}_{\mathsf{Ch},\mathcal{A}} \approx 0$ *for any polynomial-time* $\mathcal{A}$*.*

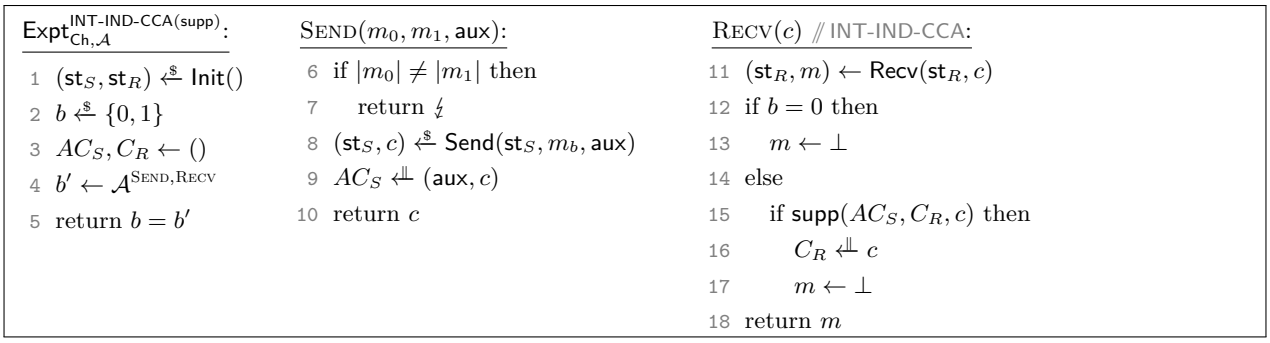

Figure 7: Experiment for INT-IND-CCA wrt. support class supp of a channel protocol Ch.

Proposition 5.12 (ROB $\wedge$ INT-IND-CCA $\Rightarrow$ ROB-INT-IND-CCA) *Let* $\mathsf{Ch} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ *be a channel,* supp *a correctness support predicate. Then for any adversary* $\mathcal{A}$ *there exist adversaries* $\mathcal{B}$ *and* $\mathcal{C}$ *with comparable run time such that*

$$\mathsf{Adv}^{\mathsf{ROB\text{-}INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ROB(supp)}}_{\mathsf{Ch},\mathcal{B}} + 4 \cdot \mathsf{Adv}^{\mathsf{INT\text{-}IND\text{-}CCA(supp)}}_{\mathsf{Ch},\mathcal{C}}.$$

Proof. Assume that we have an attacker against ROB-INT-IND-CCA. Note that the only way for $\mathcal{A}$ to get some output $m \neq \bot$ from the receiver oracle for a query c is when $b = 1$ and

- the ciphertext is supported and $m^{real} \neq m^{corr}$, or
- the ciphertext is unsupported, in which case $m^{corr} = \bot$, and we then have $m^{real} \neq \bot$.

Note that one of the two cases must happen first. We first show that if this is the first case then we can break robustness of the channel protocol. The second case will be covered by the INT-IND-CCA property which only overwrites the message for supported ciphertexts.

For the first case note that all queries of $\mathcal{A}$ to the receiver oracle up to the point where it submits an supported ciphertext c yielding $m^{real} \neq m^{corr}$ return $\bot$. We argue that this cannot happen too often by the robustness of the channel protocol. We can therefore simulate $\mathcal{A}$ through an adversary $\mathcal{B}$ playing the robustness game. Algorithm $\mathcal{B}$ first picks a random bit b and answers $\mathcal{A}$'s oracle queries (aux, m_0, m_1) to Send by checking that $|m_0| = |m_1|$, returning $\lightning$ if not, and otherwise forwarding (aux, m_b) to its own Send oracle. Adversary $\mathcal{B}$ returns the oracle's reply to $\mathcal{A}$. To simulate the receive oracle $\mathcal{B}$ replies to each query c of $\mathcal{A}$ with $\bot$ if $b = 0$, and otherwise forwards the query to its own Recv oracle, but returns $\bot$ to $\mathcal{A}$.

The simulation through $\mathcal{B}$ is perfect up to the submission of $\mathcal{A}$'s supported ciphertext c in question, because we assume that all queries to Recv before return $\bot$. For query c attacker $\mathcal{B}$ then causes its experiment to satisfy the if-clause $m^{real} \neq m^{corr}$ in Line 15 in the robust experiment in Figure 3. This sets win to true and thus makes $\mathcal{B}$ break robustness.

If the first query in $\mathcal{A}$'s attack to Recv returning a message different from $\bot$ is for an unsupported ciphertext c, then it holds that $m^{real} \neq \bot$. We can now run a black-box simulation $\mathcal{C}$ of $\mathcal{A}$, where $\mathcal{C}$ answers each Recv call with $\bot$ but forwards the query to its own oracle. If at some point $\mathcal{C}$ receives a reply distinct from $\bot$ in one of such queries then it immediately outputs 1, else it eventually outputs a random bit. An analysis similar to the one of Proposition 5.10 shows that $\mathcal{C}$ succeeds with an advantage of at least $\frac{1}{4}$ times the probability that $\mathcal{A}$ wins with an unsupported ciphertext. □

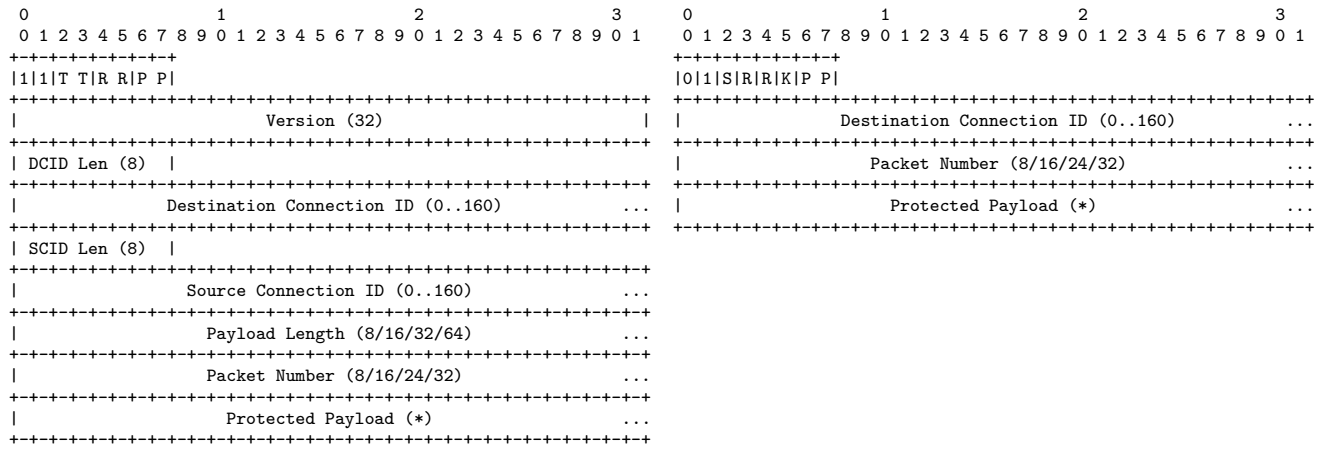

Figure 8: QUIC packet formats for long (left) and short (right) packets [13, Section 17]. The first byte contains flags T: Type, R: Reserved, P: Packet number length, S: Spin, K: Key phase. The protected payload is of variable length, indicated by `(*)`.

6 QUIC

QUIC was initially designed and implemented by Google. Currently, QUIC is in the process of being standardized by the IETF [13, 27].

QUIC distinguishes a variety of different packet types, mostly following either a long or short packet format [13, Section 17]. For reference, we illustrate both formats in Figure 8, but for the purpose of our analysis focus on the short packet format, which in particular is used for sending the main application data.

6.1 QUIC Encryption Specifications

In the following, we provide a brief overview of the encryption specifics of QUIC. QUIC packets consist of a header and a payload, the latter being encrypted using an AEAD scheme. For this encryption, the packet number forms the AEAD nonce (with a per-session random offset), and the unprotected header is used as the associated data. Headers in particular contain between 1 and 4 bytes of the packet number, allowing the receiver to reconstruct the correct nonce of (possibly reordered) packets within an appropriately-sized sliding window.

After packet encryption, QUIC additionally applies a header protection mechanism based on one of the nonce-hiding AE constructions proposed by Bellare et al. [5], and further allows keys to be updated during the channel's lifetime. Delignat-Lavaud et al. [10] treat the header protection mechanism in their analysis of the QUIC protocol, and we defer the interested reader to their paper as well as the specification [13]. Günther and Mazaheri [12] introduced a security model for multi-key channels allowing key updates, but treating only reliable channels like TLS. In our analysis of QUIC, we do not treat header protection or key updates. We argue that our results still provide reasonable insights into the robustness of the QUIC channel, if one is willing to assume that header protection (happening after our sending, resp. before our receiving steps) and key updates (corresponding to a sequence of robust channels per phase) work as intended. Confirming these assumptions and analyzing the QUIC channel in a model treating all these aspects is left as an avenue for future work.

6.2 QUIC as a Channel Protocol

When capturing QUIC as a cryptographic channel protocol, the first question arising is which interfaces to higher- and lower-level protocols should be considered. The lower-level interface is simple: Running over UDP, QUIC outputs distinct (atomic) chunks of ciphertexts accompanied by headers in a datagram-oriented manner.

For the higher-level interface, things are less clear: While QUIC offers a multiplexed interface of several parallel data streams to an application using it, its cryptographic packet protection merely works on atomic chunks of payload data which results from QUIC-internal, higher-level multiplexing and other processing.

The focus of this work being robustness of channels, we restrict ourselves to the core cryptographic packet protection mechanism of QUIC which handles robustness in transmission over the underlying UDP protocol. This means we do not consider meta-information (like handling connection identifiers), handling of multiplexed data or the option to switch encryption keys (see [12, 20] for treatments of channel notions aiming at the latter two); we accordingly consider a restricted packet header. Note that this still goes beyond the basic AEAD encryption process itself. In particular, we treat the parsing process of QUIC packet headers which play a crucial role for robustness in determining which packets can (still) be correctly received within a reordered sequence.

6.2.1 Construction

We capture QUIC as the channel protocol $\mathsf{Ch}_{\mathsf{QUIC}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ described in Figure 9. It is built from any AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ (as defined in Section 2.2) with associated key space $\mathcal{K}$ and error symbol $\perp$, the latter being inherited by the construction. QUIC employs a dynamic sliding window with an anti-replay window (for some scheme-dependent fixed replay window size w_r), i.e., at the support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$ as defined in Section 3.2. QUIC's sliding window is set dynamically on the sender side, spanning 1–4 bytes wide around the next expected packet number pn_R (i.e., the one subsequent to the highest successfully received packet number), where pn_R is the rightmost entry in the left half of the window. We formalize this through an auxiliary information space $\mathcal{X} = \{(2^4 - 1, 2^4), (2^8 - 1, 2^8), (2^{12} - 1, 2^{12}), (2^{16} - 1, 2^{16})\}$ corresponding to 8, 16, 24, and 32 bit wide windows respectively, with (almost) half-sized $w_b + 1 = w_f$.[4]

Packet numbers play a crucial role for the sliding-windows technique in QUIC, and hence also in the construction. As described in Section 6.1, QUIC packet numbers determine the nonce and also (partially) the associated data for the AEAD scheme. Packet numbers are a running integer counter on the sender's side in the range from 0 to $2^{62} - 1$. QUIC then derives the nonce for packet encryption as the XOR of a (static) initialization vector IV (a 96-bit value obtained through key generation) and the packet number (accordingly padded with 0-bits). In our construction, this translates to sampling IV at random upon channel initialization and deriving the sending nonce based on a running sending counter pn_S. While QUIC puts various header information in its packets (which enters the AEAD encryption as associated data), we focus here only on the partial, encoded packet number epn. Upon sending, epn is derived as the last n bits (for a dynamic sliding window size n) of the sending packet number pn_S. Upon receiving, epn (of length n) is decoded to the (unique) packet number matching epn in its last n bits number which is contained in the 2^n-sized window centered around the next expected packet number pn_R [13, Appendix A]. We capture these encoding and decoding steps through the following sub-algorithms, and illustrate decoding within a sliding window in Figure 10 as follows:

[4] Recall that in the formalization of $\mathsf{supp}_{dw\text{-}r[w_r]}$, the next expected packet index n is always contained in the dynamic window, hence the backwards window reaches back only $n/2 - 1$ positions for an n-sized window.

$\underline{\mathsf{Init}()}$:

1 $K \xleftarrow{\$} \mathcal{K}$
2 $IV \xleftarrow{\$} \{0,1\}^{96}$
3 $pn_S \leftarrow pn_R \leftarrow 0$ // next packet number to be sent/received
4 $R \leftarrow 0^{w_r+1}$ // replay-check bitmap of received ciphertexts in w_r-sized window
5 $\mathsf{st}_S \leftarrow (K, IV, pn_S)$
6 $\mathsf{st}_R \leftarrow (K, IV, pn_R, R)$
7 return $(\mathsf{st}_S, \mathsf{st}_R)$

$\underline{\mathsf{Send}(\mathsf{st}_S, m, \mathsf{aux})}$:

8 parse st_S as (K, IV, pn_S)
9 if $pn_S \geq 2^{62}$ then return $(\mathsf{st}_S, \perp)$ // exceeded PN space
10 $epn \leftarrow \mathsf{Encode}(pn_S, \mathsf{aux})$
11 $N \leftarrow IV \oplus pn_S$
12 $AD \leftarrow epn$
13 $c' \leftarrow \mathsf{Enc}(K, N, AD, m)$
14 $c \leftarrow (epn, c')$
15 $pn_S \leftarrow pn_S + 1$
16 $\mathsf{st}_S \leftarrow (K, IV, pn_S)$
17 return (st_S, c)

$\underline{\mathsf{Recv}(\mathsf{st}_R, c)}$:

18 parse st_R as (K, IV, pn_R, R)
19 parse c as (epn, c')
20 $pn \leftarrow \mathsf{Decode}(epn, pn_R)$ // decode pn wrt. next expected packet number
21 $N \leftarrow IV \oplus pn$
22 $AD \leftarrow epn$
23 $m \leftarrow \mathsf{Dec}(K, N, AD, c')$
24 if $m = \perp$ // AEAD decryption error
25 or $pn < pn_R - 1 - w_r$ // older than replay-check window
26 or $(pn < pn_R$ and $R[pn - pn_R + w_r + 2] = 1)$ // replay
27 return $(\mathsf{st}_R, \perp)$ // reject
28 if $pn < pn_R$ then // pn within replay window
29 $R[pn - pn_R + w_r + 2] \leftarrow 1$ // mark pn as received
30 else // pn beyond replay window
31 $R \leftarrow R \ll (pn - pn_R + 1)$ // shift window
32 $R[w_r + 1] \leftarrow 1$ // mark pn as received (last entry in window)
33 $pn_R \leftarrow pn + 1$ // set next expected pn
34 $\mathsf{st}_R \leftarrow (K, pn_R, R)$
35 return (st_R, m)

Figure 9: The abstract $\mathsf{Ch}_{\mathsf{QUIC}}$ channel protocol based on a generic AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$.

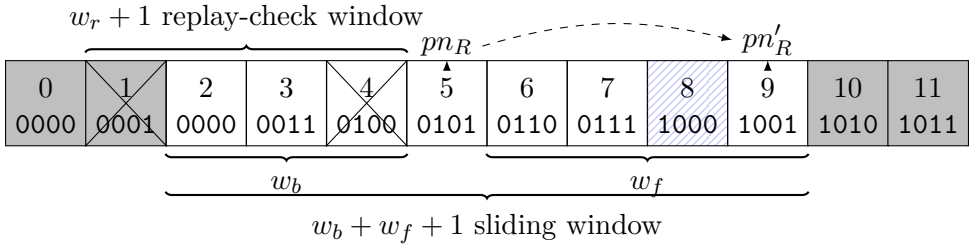

Figure 10: Exemplary illustration of a dynamic sliding receiving window of (toy) size $2^n = 8$ (i.e., $w_b = 3$ and $w_f = 4$) around the next expected packet number $pn_R = 5 = \mathtt{0101}_2$, replay-check window of size $w_r + 1 = 4$. Packet numbers 1 and 4 have been received before, crossed-out in the replay-window. Grayed-out packet numbers are outside the current sliding window.
In this situation, a received partial packet number $epn = \mathtt{000}_2$ will be (uniquely) decoded to $pn = 8 = \mathtt{1000}_2$ within the window (marked with diagonal lines), leading pn_R to be updated to $pn'_R = 9$ and also move the two windows forward.

$\underline{\mathsf{Encode}(pn_S, \mathsf{aux})}$:

1 parse aux as $(2^{n/2}-1, 2^{n/2})$
2 return $pn_S[62-n..62]$ // n-bit string

$\underline{\mathsf{Decode}(epn, pn_R)}$:

1 $n \leftarrow |epn|$
2 return $pn \in [0, 2^{62}-1]$ s.t.
$pn[62-n..62] = epn$ and
$pn_R - 2^{n-1} < pn \leq pn_R + 2^{n-1}$

In more detail, the construction works as follows.

Init. The initialization algorithm starts with sampling a key K uniformly at random from the key space $\mathcal{K}$ of the AEAD scheme, as well as a random (static) initialization vector IV of 96 bits length. The sending and receiving state, beyond K and IV, contain counters for the *next* packet number to be sent pn_S, resp. to be received pn_R, initialized to 0. Furthermore, the receiving state holds a (initially all-zero) bitmap R of size $w_r + 1$ later used to record previously seen packet numbers in a window of size w_r before the last successfully received packet number (+1 to account for the latter, too).

Send. The sending algorithm first ensures that the sending packet number pn_S does not exceed the maximal value of $2^{62}-1$. It derives the encoded packet number epn to be transmitted as the least significant 1–4 bytes of pn_S, captured through the Encode algorithm given above. It then computes the packet encryption nonce N as the XOR of the static IV and the running packet number pn_S (implicitly padded to a 96-bit bitstring). The ciphertext c' is computed as the AEAD-encryption of the input message m, using N as nonce and epn as associated data. The encoded packet number epn together with c' form the full ciphertext c. The final output is the sending state, with the packet number incremented, together with c.

Recv. The receiving algorithm begins with decoding the encoded packet number epn in the ciphertext to the full packet number pn within the dynamic sliding window around pn_R determined by $|epn|$; captured in the Decode algorithm given above. It then AEAD-decrypts the ciphertext c' using $N = IV \oplus pn$ as nonce and epn as associated data, rejecting if this step fails. The algorithm also rejects if pn is older than what is represented in the replay-check window (of w_r positions before the last successfully received packet number $pn_R - 1$) and hence cannot be ensured to not be replayed. Finally, it rejects if pn has been processed previously (determined by the bitmask R being 1 at the position corresponding to pn). Otherwise, R is marked with a 1 at the position corresponding to pn, possibly shifted before in case pn is greater than the previously highest received packet number, and the updated state and message m is output.

6.2.2 Correctness

To establish correctness wrt. support class $\mathsf{supp}_{dw\text{-}r[w_r]}$ (as defined in Section 3.2), we have to show that the QUIC channel has unique ciphertexts and correctly receives messages in supported ciphertexts, with all but small probability. For the former, we can argue that ciphertexts are unique up to a quadratic bound in the number of sent packets, assuming authenticity of the underlying AEAD scheme. We can bound the probability of a ciphertext collision happening in Send by a reduction $\mathcal{B}$ to auth of AEAD as follows. The reduction guesses when the collision happens (with $1/q_S$ probability, where q_S is the number of sent packages) as well as with which prior ciphertext this ciphertext collides (again with $1/q_S$ probability), then outputs the nonce and associated data from the former together with the AEAD ciphertext of the latter as forgery. When guessing correctly, this constitutes a valid forgery and hence ciphertext collisions in QUIC can be bounded by $q_S^2 \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{B}}$.[5]

[5] It should be noted that QUIC will be able to correctly receive even non-unique ciphertexts (within the supported windows), i.e., this bound arises as an artifact of $\mathsf{supp}_{dw\text{-}r[w_r]}$ being defined assuming unique ciphertexts (cf. Section 3.2).

To argue that QUIC correctly receives messages in ciphertexts (according to support class $\mathsf{supp}_{dw\text{-}r[w_r]}$) let us first observe the following property of QUIC's nonce encoding, which we denote as "correct decodability:" For any expected next packet number to be received $pn_R \in [0, 2^{62}-1]$, sliding window $(w_b, w_f) \in \mathcal{X}$, and (sending) packet number $pn_S \in [pn_R - \min(w_b, w_r+1), pn_R + w_f]$, it holds that

$$\mathsf{Decode}(\mathsf{Encode}(pn_S, \mathsf{aux}), pn_R) = pn_S.$$

This is achieved in QUIC by interpreting the encoded packet number in a window of size the encoded number's length ($(w_b+1+w_f) \in \{8, 16, 24, 32\}$, in bits) [13, Appendix A], while dropping packets outside of the replay window w_r before the last successfully received packet.

In order to violate correct receipt, an adversary needs to invoke Recv on a supported ciphertext (i.e., $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c)$ needs to be true in Line 15 of Figure 2) such that c decrypts to a different message than was sent. The support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$ ensures that the sent index of a ciphertext (corresponding to pn_S+1, as QUIC packet number begins with 0) is in the interval $[\mathsf{n} - \min(w_b^c, w_r+1), \mathsf{n}+w_f^c]$, where (w_b^c, w_f^c) is the auxiliary information from the Send call and n is the next expected index (corresponding to pn_R+1). QUIC's correct decodability then ensures that the decoded packet number pn equals the pn_S value used within the call to Send that output c. Hence, as $AD = epn$ and c' is part of c, Recv invokes AEAD decryption Dec on c' with the same nonce and associated data as in the corresponding encryption step in Send. By correctness of the AEAD scheme, the decrypted message will hence always equal the sent message, and thus $\mathsf{Adv}^{\mathsf{correct}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}} = 0$ for any adversary $\mathcal{A}$.

6.3 Robust Security of the QUIC Channel Protocol

We finally turn towards the security analysis of QUIC, taking its robust handling of the underlying unreliable network into account. As we will show, QUIC achieves robust confidentiality and integrity (according to the combined notion ROB-INT-IND-CCA), receiving ciphertexts within a dynamic sliding window and with a window-based replay protection; i.e., formally wrt. a support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$ (cf. Section 3.2). Compared to secure channels over reliable transports (like TLS over TCP), the security bound is not tight but, at its core, contains a loss linear in the number of received ciphertexts. After all, this is not surprising, as the adversary is given multiple tries to attack the underlying AEAD scheme, whose lower-level authenticity notion itself only considers a single forgery attempt. This result matches the observation in practice that vulnerabilities in a channels underlying encryption schemes are easier to exploit in channels over non-reliable networks (like DTLS and QUIC) than in reliable-network channels (like TLS); see, e.g., the Lucky Thirteen attack on the (D)TLS record protocols [1].[6]

Leveraging the relations between notions, we separately establish robust integrity as well as indistinguishability under chosen-plaintext attacks, yielding the combined robust confidentiality and integrity guarantees via Proposition 5.9.

Theorem 6.1 (Robust Integrity of QUIC) *Let* $\mathsf{Ch}_{\mathsf{QUIC}}$ *be the channel construction from Figure 9 from an AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$*, and support predicate* $\mathsf{supp}_{dw\text{-}r[w_r]}$ *be defined as in Section 3.2. Let* $\mathcal{A}$ *be an adversary against* $\mathsf{Ch}_{\mathsf{QUIC}}$ *in the robust integrity experiment* $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ *from Figure 4 making* q_S *queries to* Send *and* q_R *queries to* Recv*. There exists an adversary* $\mathcal{B}$ *(described in the proof) against the authenticity of* AEAD *that makes* q_S *queries to its encryption oracle* Enc *such that*

$$\mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}} \leq (q_S^2 + q_R) \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{B}}.$$

[6] The robust integrity bound for QUIC also includes a loss quadratic in the number of sent ciphertext, which is inherited from the correctness bound ruling out non-unique ciphertexts, as discussed above. We stress that this part of the bound is rather an artifact of requiring unique ciphertext for cleaner correctness support predicates (see the discussion in Section 3.2) rather than actually related to the security of QUIC. In contrast, the linear degradation in the number of received ciphertexts indeed corresponds to attacks becoming easier due to the robust treatment of reordered ciphertexts [1].

Proof. The core idea of the proof is to show that whenever the receiving oracle RECV is called in the robust integrity experiment $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ on a ciphertext c such that $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{false}$ (and hence correct receiving is skipped), we have that (a) the real receiving state st_R^{real} remains unchanged in that oracle call, and (b) the real received message is an AEAD error, i.e., $m^{real} = \bot$. This means that on unsupported ciphertexts we have $m^{real} = m^{corr} = \bot$ and since for supported ciphertexts the real and correct states stay in-sync, equality of messages also holds for such queries.

We show the above two properties by arguing that $\mathsf{Recv}(\mathsf{st}_R^{real}, c)$, in Line 24 of Figure 9, for such a call to RECV always runs into an AEAD decryption error, hence returning (a) unchanged receiving state and (b) an error output, as claimed. Having shown (b), the adversary cannot win anymore on input a non-supported ciphertext, as $m^{real} = m^{corr} = \bot$ in Line 46 of experiment $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ in this case. Furthermore (a), st_R^{real} remaining unchanged on non-supported ciphertexts, implies that in any query to RECV on a supported ciphertext, $\mathsf{st}_R^{real} = \mathsf{st}_R^{corr}$ in the two calls to Recv in Lines 40 and 43. Due to Recv being deterministic, this implies that $m^{real} = m^{corr}$ always holds in Line 46, preventing $\mathcal{A}$ from winning.

Let us first observe that a ciphertext c input to RECV can be unsupported (i.e., $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{false}$) for three reasons:

1. The input c is replayed ($c \in C_R$) or older than the replay-check window ($\mathsf{index}(c, C_S) < \mathsf{n} - w_r - 1$). In this case, the combined checks in Lines 25 and 26 of QUIC's Recv algorithm ensure such replays are detected (via the replay window, and checks of being beyond that), resulting in unchanged state and error message being output.

2. The input c is a genuine ciphertext ($(\mathsf{aux}, c) \in AC_S$ for some $\mathsf{aux} = (w_b^c, w_f^c) = (2^{n/2} - 1, 2^{n/2})$), but was reordered beyond the specified dynamic sliding window of size n, i.e., $pn_S \leq pn_R - 2^{n-1}$ or $pn_S > pn_R + 2^{n-1}$, where pn_S is the packet number in the state st_S input to Send in the SEND call that output c, and pn_R is the next expected packet number in the state st_R input to Recv in the RECV call. Then Decode will derive a different nonce N than was used for sending $c = (epn, c')$, with N matching epn. Since sent ciphertexts do not collide (by correctness, with probability bounded by q_S^2 times the auth advantage of AEAD, see above), we know that this nonce was not used in an AEAD encryption yielding c', hence if c decrypts with N to a non-error message we can output it as an AEAD forgery (see below).

3. The input $c = (epn, c')$ is not a genuine ciphertext ($c \notin C_S$). Then either epn or c' must differ compared to any genuine ciphertext, which means that either the nonce N decoded from epn or the ciphertext c' is different compared to any prior AEAD encryption. Hence, c' together with N and $AD = epn$ would be a valid AEAD forgery if it decrypts to a non-error message (which we will use below).

We formally take advantage of cases 2) and 3) in constructing the following reduction $\mathcal{B}$ against the authenticity of AEAD. Adversary $\mathcal{B}$ simulates the robust integrity game $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ for $\mathcal{A}$ by not sampling a key K itself but using its encryption oracle to emulate the Enc calls within Send. To simulate the RECV oracle, $\mathcal{B}$ proceeds as follows: Initially, $\mathcal{B}$ sets $i \leftarrow 0$ and picks $j \in [1, q_R]$ at random. Whenever $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{true}$, then $\mathcal{B}$ accounts for pn_R changes (obtaining the packet number regularly as $\mathsf{Decode}(epn, pn_R)$) and simply outputs $\bot$. Otherwise, if c is outside of the window or a replay, $\mathcal{B}$ responds with $\bot$. Finally, if $c = (epn, c')$ is within the window and not a replay, but was reordered beyond its dynamic sliding window or was never output by SEND, then $\mathcal{B}$ increments $i \leftarrow i + 1$ and if $i < j$ responds with $\bot$, otherwise $\mathcal{B}$ stops and outputs $(IV \oplus \mathsf{Decode}(epn, pn_R), epn, c')$ as its forgery.

We first observe that $\mathcal{B}$ provides a sound simulation until the event that $c = (epn, c')$ is within the window and not a replay, $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{false}$, but $\mathsf{Recv}(\mathsf{st}_R^{real}, c)$ does not yield $(\mathsf{st}_R^{real}, \bot)$ due

to AEAD decryption resulting in a non-error (in Line 24 of Figure 9). With probability $1/q_R$, $\mathcal{B}$ correctly guesses the *first* such event. In that case, $\mathcal{B}$'s output is a valid forgery in the AEAD authenticity game, since $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \texttt{false}$ implies that $c \notin C_S$ (where $C_S = AC_S\langle 2 \rangle$), and hence c' was never output by $\mathcal{B}$'s encryption oracle on input $(IV \oplus \mathsf{Decode}(epn, pn_R), epn, \cdot)$; due to either the input nonce or the output AEAD ciphertext being different for any encryption call in SEND.

If instead the above event never occurs then $m^{real} = m^{corr}$ holds throughout the robust integrity game and $\mathcal{A}$ cannot win. Hence, $\mathcal{A}$'s advantage can be upper bounded by $q_R \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{B}}$. Adding the correctness bound incorporated in case 2) above (originating from the assumption of non-unique ciphertexts in $\mathsf{supp}_{dw\text{-}r[w_r]}$, cf. Section 3.2), we arrive at the overall bound. □

Theorem 6.2 (Confidentiality of QUIC) *Let* $\mathsf{Ch}_{\mathsf{QUIC}}$ *be the channel construction from Figure 9 from an AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$*, and support predicate* $\mathsf{supp}_{dw\text{-}r[w_r]}$ *be defined as in Section 3.2. Let* $\mathcal{A}$ *be an adversary against* $\mathsf{Ch}_{\mathsf{QUIC}}$ *in the IND-CPA experiment* $\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ *from Figure 5 making* q_S *queries to* SEND*. There exists an adversary* $\mathcal{B}$ *(described in the proof) against the* IND-CPA *security of* AEAD *that makes* q_S *queries to its encryption oracle* ENC *such that*

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

Proof. From an adversary $\mathcal{A}$ against the IND-CPA security of $\mathsf{Ch}_{\mathsf{QUIC}}$ we construct a reduction $\mathcal{B}$ to the IND-CPA security of AEAD as follows.

Adversary $\mathcal{B}$ simulates the (left-or-right) IND-CPA experiment for $\mathcal{A}$ faithfully, with the only exception that it does not pick a challenge bit b and AEAD encryption key itself. Instead, it uses its encryption oracle ENC (on the derived nonce and associated data, and the two left-or-right messages m_0 and m_1) in place of the AEAD encryption step within Send. When $\mathcal{A}$ eventually outputs a bit b' guess, $\mathcal{B}$ forwards b' as its own guess.

Having $\mathcal{B}$ perfectly simulating the $\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ experiment for $\mathcal{A}$, inheriting the challenge bit from its own IND-CPA game, we have that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{B}}$. □

7 DTLS 1.3

In the following we provide the full details on our channel construction for DTLS 1.3. We first describe the encryption specification for DTLS 1.3 and then provide the full details about the construction. In the final part, we show that our construction is ROB-INT-IND-CCA secure.

7.1 DTLS Encryption Specifications

The record layer of DTLS 1.3 is different from the one in TLS 1.3 in the sense that DTLS 1.3 adds a sequence number and an epoch. DTLS 1.3 generates either a full or a minimal DTLS ciphertext depending on the flags that are set in the header which we have illustrated in Figure 11.

Let us have a closer look at the encryption specifics in DTLS 1.3. A DTLS ciphertext consists of a (protected) header and an encrypted record which is generated using an AEAD scheme. As an input, the encryption algorithm takes (as usual) four inputs, namely the key K, the nonce N, the associated data AD, as well as the message m. The specification of DTLS 1.3 [23] details how the above inputs are derived. The (per-record) nonce [23, Section 4] is derived in a similar fashion as in TLS 1.3, i.e., one concatenates a 16-bit epoch number with a 48-bit sequence number obtaining a 64-bit record sequence number.[7] This value is then padded with zeros (from the left) up to the length of the initialization vector. Finally this

[7] Note that the epoch and sequence number are handled independently. Furthermore, the epoch is updated whenever a key-update has occurred.

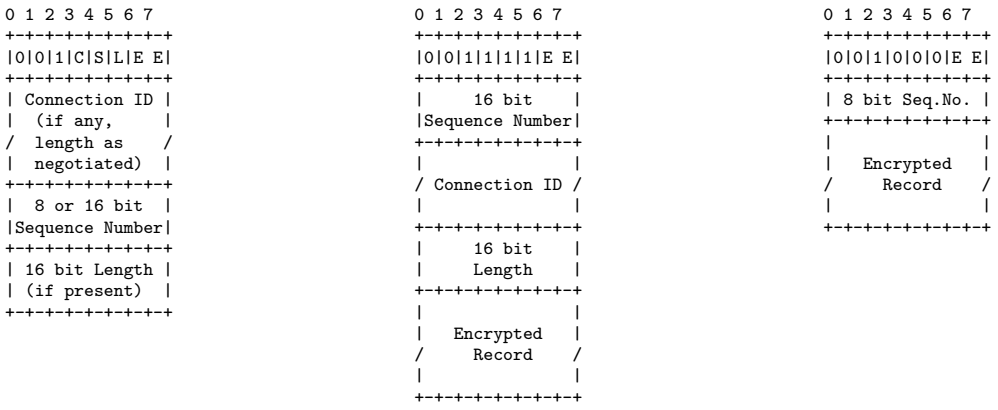

Figure 11: DTLS header types: General ciphertext header (left), and examples for full (middle) and minimal (right) DTLS 1.3 ciphertext structures [23, Section 4]. The three leftmost bits of the first byte are set to 001 indicating that the packet is a ciphertext. Furthermore, the first byte also contains flags, indicated as C: Connection ID, S: size of sequence number, L: length, E: Epoch. If the bit in C and L are set then those parts are present. In case S is set to 0 then the ciphertext structure contains an 8-bit sequence number, otherwise 16 bits. E includes the low order two bits of the epoch.

encoded sequence number is XORed with the initialization vector obtaining the nonce. The associated data is also computed similar to TLS 1.3. In more detail, in TLS 1.3 the associated data corresponds to the record header which contains information about the ciphertext data structure and the ciphertext length. DTLS 1.3 [23, Section 4] in comparison takes the same information and additionally adds the epoch and sequence number.

Another interesting aspect is that DTLS 1.3 provides a type of header protection mechanism by encrypting the record sequence number. The concept is borrowed from the QUIC specification [13] but the details are different. In DTLS 1.3 [23, Section 4.2.3], the protection mask is computed by using AEAD encryption algorithm on a sequence number key (derived from HKDF-Expand), parts of the ciphertext, and both the nonce as well as associated data being set to $\perp$. Finally, the protected sequence number is computed by XORing the leading bytes of the protection mask with the record sequence number.

Note that we do not treat key-updates and header protection in our following channel construction of DTLS 1.3. However, we argue that our results provide meaningful insights into the robustness of the DTLS 1.3 channel as long as one assumes that both the key updates and header protection function as intended. Similar to QUIC, we leave this avenue for future work to confirm these assumptions and analyze the DTLS 1.3 channel covering all of these aspects.

7.2 DTLS as a Channel Protocol

In the following, we aim to provide a cryptographic channel protocol capturing DTLS 1.3. Identical to Section 6, our focus for DTLS 1.3 is to show that our construction is indeed a robust channel. In the following, we provide a simplified version of DTLS 1.3 and the details follow.

7.2.1 Construction

We capture DTLS as the channel protocol $\mathsf{Ch}_{\mathsf{DTLS}} = (\mathsf{Init}, \mathsf{Send}, \mathsf{Recv})$ described in Figure 12. It uses an arbitrary AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$ (as defined in Section 2.2) with associated key space $\mathcal{K}$ and error symbol $\perp$, the latter being inherited by the construction. Similar to QUIC's behavior of ciphertext processing, the construction of DTLS 1.3 also employs a dynamic sliding-window technique with replay

protection window as derived from the support predicated $\mathsf{supp}_{dw\text{-}r[w_r]}$ for some scheme-dependent fixed replay window size w_r as detailed in Section 3.2. The sliding window is also set dynamically on the sender side which is spanned around the next expected sequence number sn_R and has a size of either 1 or 2 bytes. Note that the expected sequence number corresponds to the largest successfully received sequence number (on the receiving side) plus one modeling that the channel expects that the next receiving sequence number is being incremented since a new ciphertext may be received and hence the window "moves" towards the right. We formalize this through an auxiliary information space $\mathcal{X} = \{(2^4-1, 2^4), (2^8-1, 2^8)\}$ corresponding to 8-bit and 16-bit wide windows, respectively, with (almost) half-sized limits $w_b + 1 = w_f$.

In DTLS, sequence numbers and epochs play a crucial role for the sliding-window technique. Both values are used to compute the nonce and additionally the epoch serves the purpose to keep track of key-updates, i.e., the epoch is incremented whenever a key-update has occurred. However, we do not model key-updates (as explained above) here and hence do not consider epochs explicitly in the construction and only rely on sequence numbers. Note that the concept of sequence numbers is in spirit very close to the packet numbers being used in QUIC.

As described in Section 7.1, sequence numbers are used in deriving the nonce and also (partially) the associated data for the AEAD scheme. Sequence numbers are a running integer counter on the sender's side in the range from 0 to $2^{64} - 1$. DTLS 1.3 then derives the nonce as the XOR of the initialization vector IV which is a 128-bit value obtained though key generation, and the sequence number which is accordingly padded with zeros from the left. In our construction, this translates to sampling IV at random upon channel initialization and deriving the nonce on sending based on the running sn_S counter. DTLS 1.3 includes various header information into the associated data that enters the AEAD encryption process, we limit those information for modeling purposes to solely include the sequence number. Upon sending the encrypted record, DTLS 1.3 includes in the header an encoded sequence number whose encoding is derived in the sending algorithm based on the sequence number sn_S and the dynamic sliding window size given through the auxiliary input. While receiving the ciphertext, the receiver algorithm aims to reconstruct the (full) sequence number from the encoded one which is numerically closest to the next expected sequence number sn_R (cf. [23, Section 4.2.2]). This decoding mechanism provides some opportunity for ambiguity. For example, consider a two-bit window and the received encoded sequence number corresponds to `11` and the expected sequence number is `0101` (hence a binary representation of 5). The above decoding algorithm would now aim to reconstruct the sequence number which is numerically closest but given the window size the reconstruction could now either reconstruct to `0011` (corresponding to 3) or `0111` (corresponding to 7). However, only one of them should be in the two-bit window. In case, the decoding algorithm reconstructs the wrong sequence number then DTLS 1.3 fails to re-compute the per-record nonce and hence decryption of the ciphertext fails.

In order to avoid this ambiguity, we decided to model encoding and decoding exactly as in QUIC adapting the notation and length requirements to DTLS 1.3. Therefore, we obtain the following two sub-algorithms that handle encoding and decoding respectively:

$\mathsf{Encode}(sn_S, \mathsf{aux})$:

1 parse aux as $(2^{n/2} - 1, 2^{n/2})$
2 return $sn_S[64 - n..64]$ // n-bit string

$\mathsf{Decode}(esn, sn_R)$:

1 $n \leftarrow |esn|$
2 return $sn \in [0, 2^{64} - 1]$ s.t.
$sn[64 - n..64] = esn$ and
$sn_R - 2^{n-1} < sn \leq sn_R + 2^{n-1}$

In more detail, the construction works as follows.

Init. The initialization algorithm starts with sampling a key K uniformly at random from the key space $\mathcal{K}$ of the AEAD scheme, as well as a random (static) initialization vector IV of 128 bits length. The sending and receiving state, beyond K and IV, contain sending and receiving packet numbers pn_S

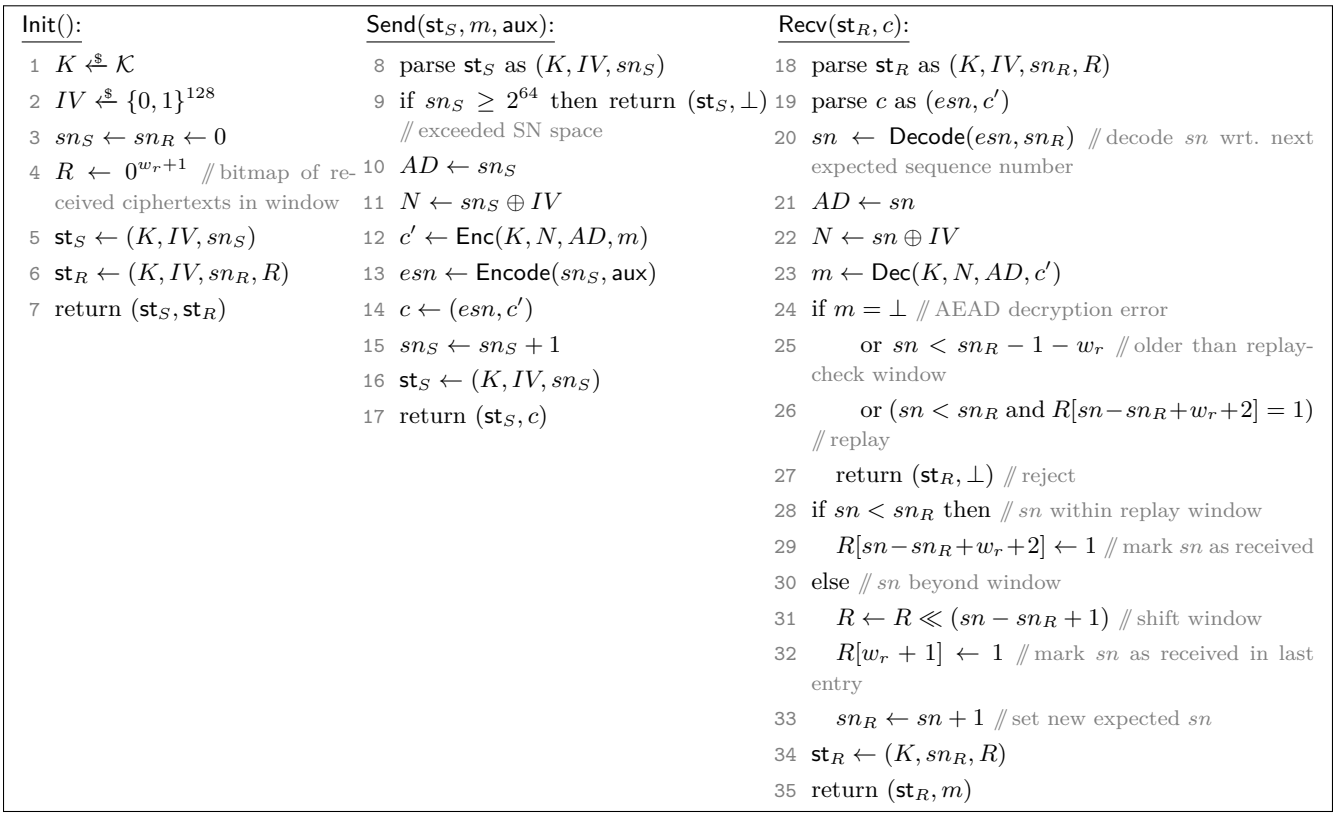

Figure 12: The abstract $\mathsf{Ch}_{\mathsf{DTLS}}$ channel protocol based on a generic AEAD scheme $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$.

and pn_R, respectively, initialized to 0. The receiving state furthermore contains a (initially all-zero) bitmap R of size $w_r + 1$ to record previously received sequence numbers and providing for later use a replay protection mechanism.

Send. The sending algorithm first ensures that the sending (record) sequence number sn_S does not exceed the maximal value of $2^{64} - 1$. It then sets this sequence number to correspond to associated data. Then it continues computing the per-record nonce N as the XOR of the sequence number (implicitly padded to a 128-bit string) with the initialization vector. The ciphertext c' is the computed as the AEAD-encryption of the input message m, using N as nonce and sn_S as associated data. Next it derives the encoded sequence number esn as the least 8 or 16 bits of sn_S which is captured by running the above Encode algorithm. The full ciphertext c is then formed as the pair consisting of encoded sequence number tsn and the AEAD ciphertext c'. The final output is the sending state, with the sequence number incremented, together with c.

Recv. The receiving algorithm begins with decoding the encoding sequence number in the ciphertext to the full sequence number sn within the dynamic sliding window centered around sn_R and determined through the length of esn which we capture by running the above decoding algorithm Decode. In order to avoid timing attacks, the algorithm first prepares the required inputs to perform the AEAD decryption algorithm and only checks afterwards if the sequence number is valid ensuring that no replay has occurred. In more detail, the algorithm rejects if the AEAD decryption failed, or if the received sequence number is older than (and hence before) the current replay window, or if the sequence number has indeed been previously processed which is determined by checking whether R contains a bit 1 at the respective position of the sequence number. Otherwise, if the previous checks were successful then R is marked with 1 at the corresponding position of sn (either directly or after shifting the replay window in case sn is greater than the previously highest received sequence number sn_R). The final output is the receiving state, with the sequence number being incremented, and the successfully decrypted message m.

7.2.2 Correctness

In order to argue correctness for the DTLS channel construction wrt. support class $\mathsf{supp}_{dw\text{-}r[w_r]}$, we first need to argue that the DTLS 1.3 channel has unique ciphertexts and in a second step we show that the channel receives messages correctly from supported ciphertexts with all but small probability. From an adversary's point of view, in order to break correctness the adversary needs to set the winning flag win to 1 in either the SEND or RECV oracle.

We begin with the former case and argue that ciphertexts are unique up to a quadratic bound in the number of sent records assuming authenticity of the underlying AEAD scheme. In more detail, to break correctness through the SEND oracle, the adversary is required to send a fresh message, nonce and associated data such that the evaluation of Send outputs a colliding ciphertext that causes Line 7 in Figure 2 to be satisfied and hence setting the flag win to true. We leverage this by constructing an adversary $\mathcal{B}$ against the authenticity game of AEAD. This adversary simulates the correctness game for $\mathcal{A}$ and does not sample an own key but instead uses its own encryption oracle in order to emulate the encryption calls within Send. Adversary $\mathcal{B}$ stores each received ciphertext from the encryption oracle with the corresponding input tuple (N, AD, m), i.e., $\mathbf{M}[c] = (N, AD, m)$. To simulate the SEND oracle, $\mathcal{B}$ proceeds as follows: $\mathcal{B}$ picks $i, j \in [1, \ldots, q_S]$ at random with the condition $i \neq j$. Whenever $\mathcal{A}$ sends a message query then $\mathcal{B}$ generates appropriately the nonce and associated data and sends the tuple (N, AD, m) to its encryption oracle relaying the answer c back to $\mathcal{A}$. Finally, after $\mathcal{A}$ has made q_S many queries, $\mathcal{B}$ outputs (N_i, AD_i, c_j) as its forgery determined using its map $\mathbf{M}$ and its chosen indices.

First we observe that $\mathcal{B}$ provides a sound simulation until a collision occurs. With probability $1/q_S$, $\mathcal{B}$ has guessed the correct query index j in which the collision occurs and also with the same probability it has guessed the correct query with which the query has collided. If the above guesses are correct, then $\mathcal{B}$ has formed a valid forgery which decrypts correctly and this ciphertext has not been generated using the respective nonce and associated data from the forgery. Hence, overall we can bound the adversary's advantage in breaking correctness by $q_S^2 \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{B}}$.

For the second case, i.e., DTLS 1.3 correctly receives messages from ciphertexts wrt. to the support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$, let us first observe that we require the same property as in QUIC about the nonce encoding for the AEAD scheme, namely correct decodability (cf. Section 6.2.1). In more detail, we require that for any next expected sequence number to be received $sn_R \in [0, 2^{64}-1]$, any sliding window $(w_b, w_f) \in \mathcal{X}$, and any sequence number $sn_S \in [sn_R - \min(w_b, w_r + 1), sn_R + w_f]$, it holds that

$$\mathsf{Decode}(\mathsf{Encode}(sn_S, \mathsf{aux}), sn_R) = sn_S.$$

The above construction of DTLS achieves this property by interpreting the encoded sequence number within a window of the sequence number's length, i.e., $(w_b + 1 + w_f) \in \{8, 16\}$. Furthermore, any package containing a sequence number which is outside of the replay window will be discarded.

In order to violate correct receipt of a message, the adversary invokes Recv on a supported ciphertext (i.e., $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c)$ such that Line 15 of Figure 2) is true meaning that c decrypts to a different message than the one that was originally sent. The given support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$ ensures that the sent index of a ciphertext is in the interval $[\mathsf{n} - \min(w_b^c, w_r + 1), \mathsf{n} + w_f^c]$, where (w_b^c, w_f^c) is the auxiliary information from the Send call and n is the next expected index (corresponding to $sn_R + 1$). DTLS' correct decodability property ensures that the decoded (full) sequence number sn equals the sn_S sequence number used within the call to Send that output c. Hence, as $AD = sn$ and c' is part of c, Recv invokes AEAD decryption Dec on c' with the same nonce and associated data as in the corresponding encryption step in Send. By correctness of the AEAD scheme, the decrypted message will hence always equal the sent message, and thus the adversary has no further advantage in breaking correctness.

7.3 Robust Security of the DTLS Channel Protocol

In the following we turn our focus towards analyzing the security of DTLS 1.3. In more detail, we wish to show on the one hand that our above channel construction from Figure 12 achieves robust integrity for the support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$. Additionally, we show that this construction also achieves confidentiality for the same support predicate. Following the implication that we established in Section 5.3 with Proposition 5.9, we then finally argue that our channel construction for DTLS 1.3 achieves the combined ROB-INT-IND-CCA notion.

Theorem 7.1 (Robust Integrity of DTLS) *Let* $\mathsf{Ch}_{\mathsf{DTLS}}$ *be the channel construction from Figure 12 from an AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$*, and support predicate* $\mathsf{supp}_{dw\text{-}r[w_r]}$ *be defined as in Section 3.2. Let* $\mathcal{A}$ *be an adversary against* $\mathsf{Ch}_{\mathsf{DTLS}}$ *in the robust integrity experiment* $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{DTLS}},\mathcal{A}}$ *from Figure 4 making* q_S *queries to* Send *and* q_R *queries to* Recv*. There exists an adversary* $\mathcal{B}$ *(described in the proof) against the authenticity of* AEAD *that makes* q_S *queries to its encryption oracle* Enc *such that*

$$\mathsf{Adv}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{DTLS}},\mathcal{A}} \leq (q_S^2 + q_R) \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{B}}.$$

Proof. The idea of the proof is similar to the robust integrity proof of QUIC (cf. Theorem 6.1). We start with reviewing the idea and then provide the respective details for our channel construction $\mathsf{Ch}_{\mathsf{DTLS}}$.

The main idea of the proof is to show that whenever the receiving oracle Recv is called in the robust integrity experiment $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{DTLS}},\mathcal{A}}$ on a ciphertext c such that $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \texttt{false}$

(and hence correct receiving is skipped), we have that (a) the real receiving state st_R^{real} remains unchanged in that oracle call, and (b) the real received message is an AEAD error, i.e., $m^{real} = \bot$.

Observe that for such a ciphertext call to $\mathsf{Recv}(\mathsf{st}_R^{real}, c)$, i.e., executing Line 23 of Figure 12, it calls the RECV oracle always resulting into a AEAD decryption error which is output in Line 24. This simply results in (a) outputting an unchanged state st_R^{real}, and (b) an erroneous output as claimed. Having shown (b), the adversary cannot win anymore on input of a non-supported ciphertext, as $m^{real} = m^{corr} = \bot$ in Line 46 of experiment $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{QUIC}},\mathcal{A}}$ in this case. Furthermore (a), the state st_R^{real} remains unchanged on non-supported ciphertexts which implies that in any query to RECV on a supported ciphertext, $\mathsf{st}_R^{real} = \mathsf{st}_R^{corr}$ in the two calls to Recv in Lines 40 and 43 in Figure 4. Due to Recv being deterministic, this implies that $m^{real} = m^{corr}$ always holds in Line 46, preventing $\mathcal{A}$ from winning.

We can consider three different cases where one observes that a ciphertext c as input to RECV is unsupported.

1. The ciphertext c is replayed ($c \in C_R$) or older than the replay-check window which formally means that $\mathsf{index}(c, C_S) < \mathsf{n} - w_r - 1$. In this case, the combined checks in Lines 25 and 26 of DTLS' Recv algorithm ensure that such replays are detected via the replay window, resulting in an unchanged state and error message being output.

2. The input c is a genuine ciphertext ($(\mathsf{aux}, c) \in AC_S$ for some $\mathsf{aux} = (w_b^c, w_f^c) = (2^{n/2} - 1, 2^{n/2})$), but was reordered beyond the specified dynamic sliding window of size n, i.e., more formally $sn_S \leq sn_R - 2^{n-1}$ or $sn_S > sn_R + 2^{n-1}$, where sn_S is the packet number in the state st_S input to Send in the SEND call that outputs c, and sn_R is the next expected sequence number in the state st_R input to Recv in the RECV call. Using our above decoding algorithm Decode basically boils down to deriving a different nonce N than the one used in the sending algorithm outputting $c = (esn, c')$ with N matching esn. By correctness of $\mathsf{Ch}_{\mathsf{DTLS}}$ we know that sent ciphertexts are unique, and hence we can conclude that this nonce was not used in an AEAD encryption yielding c'. Therefore, if c decrypts with N to a non-erroneous message we can output it as an AEAD forgery.

3. The input $c = (esn, c')$ is not a genuine ciphertext ($c \notin C_S$). In this case, either esn or c' must differ compared to any genuine ciphertext. This basically means that either the nonce N derived from the decoded epn (XORed with IV) or the ciphertext c' is different compared to any prior AEAD encryption. Hence, c' together with N and $AD = sn$ (derived from the decoded epn) would be a valid AEAD forgery if it decrypts to a non-erroneous message.

In the following, we use cases 2) and 3) to construct a reduction $\mathcal{B}$ against the authenticity of AEAD. Adversary $\mathcal{B}$ simulates the robust integrity game $\mathsf{Expt}^{\mathsf{ROB\text{-}INT}(\mathsf{supp}_{dw\text{-}r[w_r]})}_{\mathsf{Ch}_{\mathsf{DTLS}},\mathcal{A}}$ for $\mathcal{A}$ by not sampling a key K itself but using its encryption oracle to emulate the Enc calls within Send. To simulate the RECV oracle, $\mathcal{B}$ proceeds as follows: Initially, $\mathcal{B}$ sets $i \leftarrow 0$ and picks $j \in [1, q_R]$ uniformly at random. Whenever the ciphertext is supported, i.e., $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{true}$, then $\mathcal{B}$ accounts for sn_R to change (obtaining the sequence number as usual via $\mathsf{Decode}(esn, sn_R)$) and simply outputs $\bot$. Otherwise, if c is outside of the window or a replay, $\mathcal{B}$ responds with $\bot$. Finally, if $c = (esn, c')$ is within the window and not a replay, but was reordered beyond its dynamic sliding window or was never output by SEND, then $\mathcal{B}$ increments $i \leftarrow i+1$ and if $i < j$ responds with $\bot$, otherwise $\mathcal{B}$ stops and outputs $(IV \oplus \mathsf{Decode}(esn, sn_R), esn, c')$ as its forgery.

We first observe that $\mathcal{B}$ provides a sound simulation until the event that $c = (esn, c')$ is within the window and not a replay, $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{false}$, but $\mathsf{Recv}(\mathsf{st}_R^{real}, c)$ does not output $(\mathsf{st}_R^{real}, \bot)$ due to AEAD decryption resulting in a non-error (in Line 24 of Figure 12). With probability $1/q_R$, $\mathcal{B}$ correctly guesses the *first* such event. In that case, $\mathcal{B}$'s output is a valid forgery in the AEAD authenticity game, since $\mathsf{supp}_{dw\text{-}r[w_r]}(AC_S, C_R, c) = \mathtt{false}$ implies that $c \notin C_S$ (where $C_S = AC_S\langle 2 \rangle$), and hence c' was

never output by $\mathcal{B}$'s encryption oracle on input $(IV \oplus \mathsf{Decode}(esn, pn_R), esn, \cdot)$; due to either the input nonce or the output AEAD ciphertext being different for any encryption call in SEND.

If instead the above event never occurs then $m^{real} = m^{corr}$ holds throughout the robust integrity game and $\mathcal{A}$ cannot win. Hence, $\mathcal{A}$'s advantage can be upper bounded by $q_R \cdot \mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD},\mathcal{B}}$. From our assumption of non-unique ciphertexts in the support predicate $\mathsf{supp}_{dw\text{-}r[w_r]}$, we need to additionally incorporate our correctness bound due to the second case from above. Note that this additional bound is rather an artifact from our model of requiring unique ciphertexts in order to have cleaner support predicates.

Taking the above into account, we arrive at the overall bound. □

Theorem 7.2 (Confidentiality of DTLS) *Let* $\mathsf{Ch}_{\mathsf{DTLS}}$ *be the channel construction from Figure 12 from an AEAD scheme* $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$*, and support predicate* $\mathsf{supp}_{dw\text{-}r[w_r]}$ *be defined as in Section 3.2. Let* $\mathcal{A}$ *be an adversary against* $\mathsf{Ch}_{\mathsf{DTLS}}$ *in the IND-CPA experiment* $\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch}_{\mathsf{DTLS}},\mathcal{A}}$ *from Figure 5 making* q_S *queries to* SEND*. There exists an adversary* $\mathcal{B}$ *(described in the proof) against the* IND-CPA *security of* AEAD *that makes* q_S *queries to its encryption oracle* ENC *such that*

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

Proof. Assume that $\mathcal{A}$ is an adversary attacking $\mathsf{Ch}_{\mathsf{DTLS}}$ in the IND-CPA sense. Then we construct a new adversary $\mathcal{B}$, running $\mathcal{A}$ as a sub-routine, attacking the IND-CPA security of AEAD.

Adversary $\mathcal{B}$ simulates the (left-or-right) IND-CPA experiment for $\mathcal{A}$ faithfully with the only exception that it does not sample its own key K as well as does not pick the challenge bit b. To simulate the SEND oracle, $\mathcal{B}$ proceeds as follows. It performs an initialization phase where it samples at random an initialization vector IV as well as initializes the sending sequence number sn_S to 0. Furthermore, $\mathcal{B}$ prepares the nonce and associated data by setting the sequence number to correspond to the associated data, and it performs an XOR operation of the initialization vector and the (appropriately padded) sequence number obtaining the nonce. Upon receiving a message pair (m_0, m_1) from $\mathcal{A}$, $\mathcal{B}$ sends the tuple (N, AD, m_0, m_1) to its oracle. It receives back a ciphertext c'. $\mathcal{B}$ then encodes the sequence number obtaining esn which together with c' builds the full ciphertext c and it increments the sequence number. Next, it provides the ciphertext c to $\mathcal{A}$. When $\mathcal{A}$ eventually outputs a guess b', then $\mathcal{B}$ simply forwards b' as its own guess.

Note that $\mathcal{B}$ perfectly simulates the experiment $\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch},\mathcal{A}}$ for $\mathcal{A}$, inheriting the challenge bit from its own IND-CPA experiment. Thus, we have that $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{Ch}_{\mathsf{DTLS}},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD},\mathcal{B}}$. □

Using the results from the above proofs, we can conclude that by Proposition 5.9 it follows that our channel construction for DTLS 1.3 achieves ROB-INT-IND-CCA security.

Acknowledgments

Marc Fischlin and Christian Janson have been (partially) funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297. Felix Günther has been supported in part by Research Fellowship grant GU 1859/1-1 of the German Research Foundation (DFG).

References

[1] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013. (Cited on pages 6 and 25.)

[2] M. Backendal. Puncturable symmetric KEMs for forward-secret 0-RTT key exchange. Master's thesis, Lund University, 2019. (Cited on page 8.)

[3] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In *ACM CCS 2002*, pages 1–11, 2002. (Cited on page 3.)

[4] M. Bellare, T. Kohno, and C. Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, 2004. (Cited on pages 3 and 4.)

[5] M. Bellare, R. Ng, and B. Tackmann. Nonces are noticed: AEAD revisited. In *CRYPTO 2019, Part I*, pages 235–265, 2019. (Cited on page 21.)

[6] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In *CRYPTO 2017, Part III*, pages 619–650, 2017. (Cited on page 3.)

[7] A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In *EUROCRYPT 2012*, pages 682–699, 2012. (Cited on page 3.)

[8] C. Boyd, B. Hale, S. F. Mjølsnes, and D. Stebila. From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In *CT-RSA 2016*, pages 55–71, 2016. (Cited on pages 3, 4, 8, 9, and 10.)

[9] S. Chen, S. Jero, M. Jagielski, A. Boldyreva, and C. Nita-Rotaru. Secure communication channel establishment: TLS 1.3 (over TCP fast open) vs. QUIC. In *ESORICS 2019, Part I*, pages 404–426, 2019. (Cited on page 4.)

[10] A. Delignat-Lavaud, C. Fournet, B. Parno, J. Protzenko, T. Ramananandro, J. Bosamiya, J. Lallemand, I. Rakotonirina, and Y. Zhou. A security model and fully verified implementation for the IETF QUIC record layer. Cryptology ePrint Archive, Report 2020/114, 2020. https://eprint.iacr.org/2020/114. (Cited on page 21.)

[11] M. Fischlin, F. Günther, G. A. Marson, and K. G. Paterson. Data is a stream: Security of stream-based channels. In *CRYPTO 2015, Part II*, pages 545–564, 2015. (Cited on page 3.)

[12] F. Günther and S. Mazaheri. A formal treatment of multi-key channels. In *CRYPTO 2017, Part III*, pages 587–618, 2017. (Cited on pages 21 and 22.)

[13] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport – draft-ietf-quic-transport-25. https://tools.ietf.org/html/draft-ietf-quic-transport-25, 2020. (Cited on pages 3, 9, 11, 12, 21, 22, 25, and 28.)

[14] J. Jaeger and I. Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In *CRYPTO 2018, Part I*, pages 33–62, 2018. (Cited on pages 3 and 8.)

[15] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO 2012*, pages 273–293, 2012. (Cited on pages 3 and 4.)

[16] T. Kohno, A. Palacio, and J. 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 page 3.)

[17] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru. How secure and quick is QUIC? Provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231, 2015. (Cited on page 4.)

[18] G. A. Marson and B. Poettering. Security notions for bidirectional channels. *IACR Trans. Symm. Cryptol.*, 2017(1):405–426, 2017. (Cited on pages 3 and 8.)

[19] K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *ASIACRYPT 2011*, pages 372–389, 2011. (Cited on page 3.)

[20] C. Patton and T. Shrimpton. Partially specified channels: The TLS 1.3 record layer without elision. In *ACM CCS 2018*, pages 1415–1428, 2018. (Cited on page 22.)

[21] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), 2018. (Cited on pages 3 and 9.)

[22] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), 2012. (Cited on pages 9 and 10.)

[23] E. Rescorla, H. Tschofenig, and N. Modadugu. The Datagram Transport Layer Security (DTLS) Protocol Version 1.3 – draft-ietf-tls-dtls13-34. https://tools.ietf.org/html/draft-ietf-tls-dtls13-34, 2019. (Cited on pages 3, 9, 11, 12, 27, 28, and 29.)

[24] P. Rogaway. Authenticated-encryption with associated-data. In *ACM CCS 2002*, pages 98–107, 2002. (Cited on page 6.)

[25] P. Rogaway and Y. Zhang. Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In *CRYPTO 2018, Part II*, pages 3–32, 2018. (Cited on pages 3, 4, and 8.)

[26] T. Shrimpton. A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272, 2004. http://eprint.iacr.org/2004/272. (Cited on pages 4, 5, and 17.)

[27] M. Thomson and S. Turner. Using TLS to Secure QUIC – draft-ietf-quic-tls-25. https://tools.ietf.org/html/draft-ietf-quic-tls-25, 2020. (Cited on pages 3 and 21.)

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