# Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR*

Kenneth G. Paterson** and Gaven J. Watson***

Information Security Group,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX, U.K.
kenny.paterson@rhul.ac.uk, g.watson@rhul.ac.uk

**Abstract.** This paper presents a formal security analysis of SSH in counter mode in a security model that accurately captures the capabilities of real-world attackers, as well as security-relevant features of the SSH specifications and the OpenSSH implementation of SSH. Under reasonable assumptions on the block cipher and MAC algorithms used to construct the SSH Binary Packet Protocol (BPP), we are able to show that the SSH BPP meets a strong and appropriate notion of security: indistinguishability under buffered, stateful chosen-ciphertext attacks. This result helps to bridge the gap between the existing security analysis of the SSH BPP by Bellare *et al.* and the recently discovered attacks against the SSH BPP by Albrecht *et al.* which partially invalidate that analysis.

**Keywords:** SSH; counter mode; security proof

## 1 Introduction

SSH is one of the most widely used secure network protocols. Originally designed as a replacement for insecure remote login procedures which sent information in plaintext, it has since become a general purpose tool for securing Internet traffic. The current version of SSH, SSHv2, was designed in 1996, and it is this version to which we refer throughout this paper. The SSHv2 protocols are defined in a collection of RFCs [6, 17–20].

The SSH Binary Packet Protocol (BPP), as specified in [19], is the component of SSH that is responsible for providing confidentiality and integrity services to all messages exchanged over an SSH connection. It was subjected to a formal cryptographic security analysis using the methods of provable security by Bellare *et al.* [5]. Bellare *et al.* introduced a stateful security model and notion for SSH-style protocols. They also proved that several minor variants of the SSH BPP meet their security notion, given reasonable assumptions about the cryptographic primitives. In particular, they showed that, while the SSH BPP using CBC mode encryption with IV chaining (SSH-IPC) is *in*secure, the SSH BPP using either CBC mode encryption with explicit random IVs and random padding (SSH-$NPC), or counter mode encryption (SSH-CTR), is secure in their model.

However, the recent work of Albrecht *et al.* [1] has demonstrated plaintext recovery attacks against both SSH-IPC and SSH-$NPC, despite the proof of security for SSH-$NPC in [5]. The attacks in [1] exploit several features that are intrinsic to the SSH specification and to implementations, but that are not captured in the security model of [5]: firstly, the decryption process depends on the packet length field, which itself forms part of the plaintext data; secondly, data can be delivered to the decrypting party in a byte-by-byte manner by an attacker, allowing the attacker to observe the behaviour of the decrypting party after each byte is received; and, thirdly, the attacker can distinguish various kinds of decryption failure (most importantly, the attacker can tell exactly when a MAC fails to verify). As a consequence of these attacks, versions 5.2 and

* This research was supported in part by the European Commission under contract ICT-2007-216676 (ECRYPT-II). A short version of this paper is to appear in the proceedings of Eurocrypt 2010. This is the full version.
** This author supported by an EPSRC Leadership Fellowship, EP/H005455/1.
*** This author supported by an EPSRC Industrial CASE studentship sponsored by BT Research Laboratories.

higher of OpenSSH, the leading implementation of SSH, now negotiate the selection of counter mode in preference to CBC mode. This follows the recommendation of the CPNI vulnerability announcement [10]. OpenSSH versions 5.2 and higher also include specific counter-measures for CBC mode to frustrate the CBC-specific attacks of [1].

No attacks are known against the SSH BPP using counter mode, and the security model and proof for the relevant scheme SSH-CTR provided in [5] does rule out many classes of attack. Yet it is evident, in view of the attacks in [1], that the current formal security analysis of SSH-CTR in [5] is inadequate. In particular, the current analysis of SSH-CTR does not take into account the plaintext-dependent nature of the decryption process, nor the ability of the attacker to interact in a byte-by-byte manner with the decryption process. Indeed, the length field which turns out to be so critical to breaking SSH in [1] is ignored in the security analysis of [5], while it is assumed in [5] that ciphertexts are processed in an atomic fashion. Moreover, while the model of [5] does include errors arising from cryptographic processing, it does not do so in a way that accurately reflects the reality of SSH implementations such as OpenSSH – in the model of [5], any error condition leads to an identical error message, while in reality, the error type and the timing of the error can both leak to the adversary. This additional information was also exploited in the attacks of [1].

### 1.1 Our contribution

This paper aims to bridge the gap between the current security analysis of the SSH-CTR in [5] on the one hand, and the reality of the SSH specifications in the RFCs and the OpenSSH implementation of the SSH BPP using counter mode on the other. We develop a security model for the SSH BPP that extends the stateful model introduced in [5] and that is driven by our desire to more closely align the security model with the SSH specifications and the OpenSSH implementation. We focus on the OpenSSH implementation in preference to any of the many other SSH implementations available because of its widespread use [16]. A novel aspect of our security model is its ability to allow the attacker to interact with the decryption oracle in a byte-by-byte fashion, with ciphertext bytes being buffered until they can be processed. Novel aspects of our description of the SSH BPP using counter mode include its provision for plaintext-dependent decryption, and accurate modeling of all the error events that arise during decryption in the OpenSSH implementation of the SSH BPP in counter mode. We prove that the SSH BPP using counter mode is secure in our model, under standard assumptions concerning the cryptographic components used in the construction. This requires significant reworking of the security analysis for counter mode in [5] to take account of the new features of our model and our description of the SSH BPP. Our analysis is sufficient to show that the SSH BPP using counter mode is immune to the type of attacks reported in [1].

While our analysis is quite specific to the SSH BPP in counter mode, we believe that the modeling and proof techniques developed here should be much more widely applicable: all reasonably complex secure communication protocols involve handling of error and other management messages, and many such protocols allow for the adversary to interact with the decryption process in a fine-grained manner (rather than in a "ciphertext-atomic" manner). More generally, we hope that our practice-driven, provable security analysis of the SSH BPP will serve as an example to show that provable security techniques have an important role to play in analyzing protocols that are used in the real world, whilst taking into account low-level, code-oriented behaviours of the cryptographic elements of the protocols. Proofs in models of this type extend the scope of the provable security paradigm to new levels of realism, offering security guarantees that are much more meaningful in practice.

### 1.2 Related Work

The way in which decryption operates in SSH is related to a branch of cryptography called online encryption, as studied in [8, 3, 2, 11–14]. These papers all investigate blockwise adaptive attackers against the encryption process rather than against the decryption process, so that the attacker chooses plaintext blocks in an adaptive manner, seeing the corresponding ciphertext blocks one-by-one as the chosen plaintext blocks are encrypted. This approach is motivated by cryptographic applications for small devices such as smart cards which cannot store a whole ciphertext in internal buffers and which must therefore output ciphertext blocks as the buffer fills. As we will see, while the encryption process used in the SSH BPP could in principle be on-line, in practice (e.g. in the OpenSSH implementation) it is not. On the other hand, the adversary can interact with the decryption process in a block-by-block (or even byte-by-byte) manner, and, as the attacks of [1] show, can gain significant advantage by doing so. So while the online encryption literature appears to be related to our work, the link is superficial and the work is not directly applicable to the case of SSH.

### 1.3 Paper Organisation

We begin by giving a description of the SSH Binary Packet Protocol in Section 2, using this to identify the key features required in our modeling of the SSH BPP and its security. In Section 3 we define the building blocks that we use to define the SSH BPP's Encode-then-Encrypt&MAC encryption scheme. Section 4 gives the definitions of our new security models. Section 5 contains our proof of security for SSH using counter mode encryption. Section 6 presents our conclusions.

## 2 SSH Binary Packet Protocol

The SSH Binary Packet Protocol (BPP) is defined in RFC 4253 [19]. The SSH BPP provides both confidentiality and integrity of messages sent over an SSH connection using an encode-then-encrypt&MAC construction. A message is first encoded by prepending a 4 byte packet length field and 1 byte padding length field and appending a minimum of 4 bytes of random padding. The packet length field specifies the total length of the encoded message excluding the packet length field itself. This encoded message is then encrypted. There are various algorithms supported for encryption, but here, in the light of the attacks in [1], we only consider stateful counter mode encryption, as specified for SSH in RFC 4344 [6]. Since the SSH BPP is specified in a blockwise manner, SSH still appends padding even when using counter mode encryption. The final ciphertext is the concatenation of the encoded-then-encrypted message and a MAC value. The MAC value is computed over the concatenation of a 32-bit packet sequence number and the encoded (but not encrypted) message. The sequence number is not sent over the channel but is maintained separately by both communicating parties. Figure 1 illustrates the SSH BPP.

### 2.1 Modeling the SSH BPP and its Security

We now give a high-level description of the main features of our model for the SSH BPP and its security, explaining how these arise from features of the SSH BPP specification and specific implementations.

As with the model of [5], our model for the SSH BPP is a stateful one, reflecting the protocol's use of per-packet sequence numbers. We also wish to give the adversary access to encryption and decryption oracles in a left-or-right indistinguishability game. We next discuss how these oracles should be defined, with further details to follow in the sections ahead.

It can be seen from the above description that the encryption process used by SSH could be performed in an online manner (in the sense introduced in Section 1.2). However, this would at

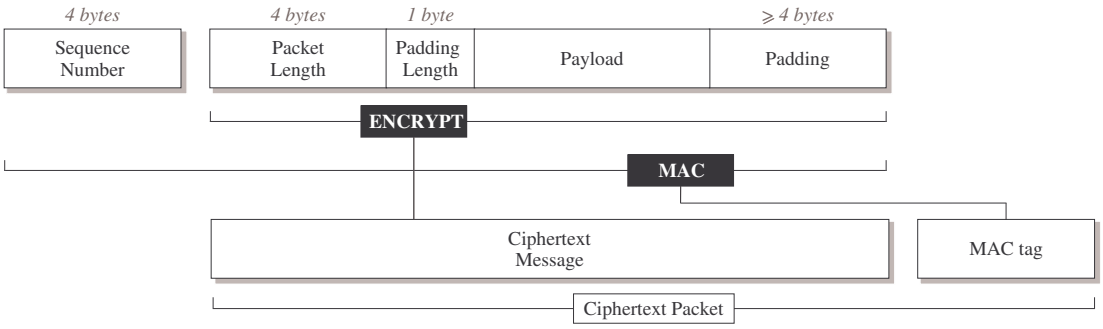


**Fig. 1.** SSH BPP packet format and cryptographic processing

least require the packet length to be fixed in advance of the first block being encrypted. In practice, the encryption is not done in an online manner. For example, in OpenSSH, the MAC on the whole plaintext is calculated prior to any encryption being performed, with the entire plaintext being buffered before this operation is carried out (see the function `packet_send2_wrapped` in the file `packet.c` of the OpenSSH source code).

At this point, our model begins to significantly diverge from the model of [5].

When decrypting a ciphertext, the receiver should first decrypt the first block received and retrieve the packet length field in order to determine how much more data must be received before the MAC tag is obtained. According to RFC 4253 [19]:

> *"Implementations SHOULD decrypt the length after receiving the first 8 (or cipher block size, whichever is larger) bytes of a packet."*

Thus we may expect that an SSH implementation will enter into a wait state, awaiting further data, unless sufficient data has already arrived to complete the packet. Informally speaking, this renders the entire decryption process plaintext-dependent, in the sense that the number of ciphertext bytes required before the decryption process can complete (possibly with an error message because of a MAC verification failure) is determined by the initial bytes of the plaintext. Moreover, because SSH is implemented over TCP, the attacker can deliver as few or as many bytes of ciphertext at a time as he wishes to the decrypting party. These facts are exploited in the attacks against the SSH BPP in CBC mode in [1]. Thus our security analysis for the SSH BPP needs to consider the length field and how its processing affects security, as well as allowing the adversary to deliver data to the decryption oracle in a byte-by-byte manner in the security model. However, it should be noted that the plaintext message is not made available to the adversary in a byte-by-byte manner as it is decrypted. Instead, in implementations, the plaintext is buffered until sufficient data has arrived that the MAC can be checked. Our model, therefore, needs to allow byte-by-byte delivery of ciphertext data, but also to include a buffered decryption process.

In fact, the situation is more complicated than this because implementations of SSH also follow the advice in RFC 4253 [19] to perform sanity checking of the length field as soon as it is obtained from the first block of ciphertext:

> *"... implementations SHOULD check that the packet length is reasonable in order for the implementation to avoid denial of service and/or buffer overflow attacks."*

What is "reasonable" is not defined in the RFCs, and specific implementations adopt various practices. Version 5.2 of OpenSSH implements a particular set of checks, and tries to tear down the SSH connection with an error message in the event that these checks fail. This error condition is generally quite easy to distinguish from a MAC failure in an attack because an SSH implementation can be made to pass through a wait state before the MAC failure. The distinguishability of

these different error conditions is used in the attacks against OpenSSH in CBC mode in [1]. So a security model for the SSH BPP should include errors arising from length checking as well as from MAC failures, and should report these errors in such a way that they can be distinguished by the adversary. Additional errors may arise after MAC checking, because of a failure of the decoding algorithm applied to the recovered, encoded message. Again, the model should reflect this possibility. To comply with the SSH specifications, all of these errors should be "fatal", leading to the destruction of the SSH connection. However, note that an adversary may be able to prevent such error messages from reaching the peer of party initiating the tear-down. We handle this aspect by having separate states for the encryption and decryption oracles in our model, and with an error arising during decryption leading to the loss of the decryption oracle, but not the encryption oracle, and vice-versa.

It is notable that SSH attempts to hide the packet length field by encrypting it. However, a simple extension of the attacks in [1] shows that this is futile: an attacker who can detect the start of a new packet simply needs to flip a bit somewhere in the ciphertext after the length field and wait for a MAC failure. Simple arithmetic involving the number of ciphertext bytes delivered before the MAC failure is seen then tells the attacker what the content of the packet length field was. Of course, the cost of this attack is to lose the SSH connection. However, it shows that the length field cannot be hidden from an active attacker. For this reason, we will insist that, in our left-or-right indistinguishability game, all pairs of messages submitted to the encryption oracle should have the same length when encoded, so that they cannot be trivially distinguished using the above attack.

## 3 Definitions

### 3.1 Notation

First let us begin by defining some notation. For a string $x$, let $|x|$ denote the length of $x$ in bytes, and let $x[i]$ denote the $i$-th block of $x$, where, throughout, blocks consist of $L$ bytes. Let $x[1\ldots n]$ denote the concatenation of the blocks $x[1], x[2], \ldots, x[n]$ of $x$ and let $x\|y$ denote the concatenation of strings $x$ and $y$. Let $\varepsilon$ denote the empty string. Let $\langle i\rangle_t$ denote the $t$-byte binary representation of integer $i$, where $0 \leq i < 2^{8t}$.

### 3.2 Building Blocks

Based on the discussion in the previous section, we now define the primitives which form the building blocks in our description of the SSH BBP's encode-then-encrypt&MAC construction. These building blocks are an encoding scheme $\mathcal{EC}$, an encryption scheme (we consider only counter mode encryption) and a message authentication scheme $\mathcal{MA}$.

**Encoding Scheme:** The *encoding scheme* $\mathcal{EC} = (\mathsf{enc}, \mathsf{dec})$ used in SSH consists of an encoding algorithm $\mathsf{enc}$ and a decoding algorithm $\mathsf{dec}$. The encoding algorithm $\mathsf{enc}$ is stateful and randomised, takes as input a message $m$ and outputs two messages $(m_e, m_t)$. Here as in [5], $m_e$ denotes the encoded message which will be used by any future encryption process and $m_t$ denotes the encoded message which will be used by a MAC tagging algorithm. As required by the SSH BPP, the encoding algorithm prepends some length information about the message and appends some padding.

The decoding algorithm $\mathsf{dec}$ is stateful and deterministic. It takes as input the full encoded message $m_e = m_e[1\ldots n]$, strips off all length fields and outputs the decoded message $m$. However, if it is unable to parse the message correctly an error message $\perp_P$ is output. Note that our definition of $\mathsf{dec}$ is slightly different to that in [5] which had two outputs $m$ and $m_t$. Note also that $\mathsf{dec}$ will

**Algorithm** $\mathsf{enc}(m)$
  **if** $st_e = \perp$ **then**
    **return** $(\perp, \perp)$
  **end if**
  **if** $SN_e \geq 2^{32}$ **or** $|m| \geq 2^{32} - 5$ **then**
    $st_e \leftarrow \perp$
    **return** $(\perp, \perp)$
  **else**
    $PL \leftarrow L - ((|m| + 5) \bmod L)$
    **if** $PL < 4$ **then**
      $PL \leftarrow PL + L$
    **end if**
    $PD \xleftarrow{r} \{0,1\}^{8 \cdot PL}$
    $LF \leftarrow (1 + |m| + PL)$
    $m_e \leftarrow \langle LF \rangle_4 \| \langle PL \rangle_1 \| m \| PD$
    $m_t \leftarrow SN_e \| m_e$
    $SN_e \leftarrow SN_e + 1$
    **return** $(m_e, m_t)$
  **end if**

**Algorithm** $\mathsf{dec}(m_e)$
  **if** $st_d = \perp$ **then**
    **return** $\perp$
  **end if**
  **if** $SN_d \geq 2^{32}$ **then**
    $st_d \leftarrow \perp$
    **return** $\perp$
  **else**
    Attempt to parse $m_e$ as:
    $\langle LF \rangle_4 \| \langle PL \rangle_1 \| m \| PD$ where
    $PL \geq 4$, $|PD| = PL$ and $|m| \geq 0$.
    **if** parsing fails **then**
      $st_d \leftarrow \perp$
      **return** $\perp_P$
    **else**
      $SN_d \leftarrow SN_d + 1$
      **return** $m$
    **end if**
  **end if**

**Fig. 2.** Encoding Scheme for SSH

only be called during the decryption process for SSH if both length checking and MAC checking have not returned errors. For correctness of the encoding scheme, we require that for any $m$ with $\mathsf{enc}(m) = (m_e, m_t) \neq (\perp, \perp)$, we have $\mathsf{dec}(m_e) \neq \perp_P$.

The specific encoding scheme used by the SSH BPP specification is shown in Figure 2. Here, $L$ denotes the block-size in bytes of the block cipher in use (or the default value of 8 if a stream cipher such as ARCFOUR is being used), $LF$ denotes the length field, $PL$ denotes the padding length and $PD$ denotes the padding bytes. The padding bytes are assumed to be random in our security analysis, though our security results also hold for any distribution on the padding bytes (including fixed bytes). We test that the message $m$ submitted for encoding contains at most $2^{32} - 6$ bytes, so that the length of the encoded message can be recorded in the 4-byte length field. Each of the two algorithms $\mathsf{enc}$, $\mathsf{dec}$ maintains a separate state of the form $(st, SN)$, initially set to $(\varepsilon, 0)$. In each case, the first component $st$ maintains the status of the algorithm, i.e. if the algorithm is in an error state or not. This is used to model the effect of an SSH connection tear-down when an error occurs. The second component $SN$ denotes a 32-bit sequence number. Note that RFC 4344 [6] states that when the sequence number $SN$ wraps around, new keys must be negotiated. For simplicity in our analysis, we model this by forcing $st_e$ (or $st_d$) to $\perp$ when $SN_e$ (or $SN_d$) reaches $2^{32}$. In our full model of the SSH BPP, this has the effect of removing the adversary's access to the encryption or decryption oracle. This ensures that each value of $SN_e$ or $SN_d$ is used only once, and is equivalent to enforcing rekeying when the relevant sequence number wraps around. Note that in [5], the equivalent state consists of a single value which is "over-loaded" to carry both the algorithm status and sequence number. For concreteness, Figure 2 shows the specific parsing steps carried out by OpenSSH during decoding. Here, it is verified that the padding length field has a value greater than 4 and that the decoded message $m$ has length zero or greater; other implementations may perform different checks here.

**Encryption Scheme:** The construction of SSH that we consider uses counter mode encryption of a block cipher, and is called SSH-CTR in [5]. When we come to formally analyze the security of SSH-CTR, we will regard the block cipher as being a pseudorandom function family rather than as a pseudorandom permutation family. This allows us to directly use some of the results from [4]. Our definition for a pseudorandom function family can be found in Appendix A.1.

Counter mode encryption $\mathrm{CTR}[F] = (\mathcal{K}\text{-CTR}, \mathcal{E}\text{-CTR}, \mathcal{D}\text{-CTR})$ consists of three algorithms, detailed in Appendix A.1. The key generation algorithm $\mathcal{K}$-CTR outputs a random $k$-bit key $K_e$ for

the underlying pseudorandom function family $F$, therefore specifying a function $F_{K_e}$ having $l$-bit inputs and $L$-byte outputs. Note that in practice we have $l = 8L$ since all block ciphers have equal input and output size. The key generation algorithm also outputs a random $l$-bit initial counter $ctr$, which is used to initialise counters in both the encryption and decryption algorithms. Because of the encoding algorithm used in encryption and the length checking used during decryption (see Section 3.3), we can assume that the encryption and decryption algorithms $\mathcal{E}$-CTR, $\mathcal{D}$-CTR both take as input a sequence of plaintext or ciphertext bytes which can be split into a sequence of $L$-byte blocks.

We also define the scheme $\mathrm{CTR}^{\mathcal{EC}}[F]$ to be a combination of counter mode encryption and the encoding/decoding scheme from Figure 2, with the detailed algorithms for this scheme appearing in Appendix A.1. This construction is not used in SSH, but is needed as a step in our security analysis in Section 5. The scheme $\mathrm{CTR}^{\mathcal{EC}}[F]$ features a buffered decryption algorithm: it takes as input a string $c$ of any length which is added to a buffer `cbuff` and then used as required. This reflects the way that buffered decryption occurs in our overall model for SSH-CTR.

**Message Authentication Scheme:** A *message authentication scheme* (MAC) $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ consists of three algorithms. The key generation algorithm $\mathcal{K}_t$ returns a key $K_t$. The tag algorithm $\mathcal{T}$, which may be stateful and randomised, takes as input the key $K_t$ and an encoded message $m_t$ and returns a tag $\tau$. The verification algorithm $\mathcal{V}$, which is deterministic and stateless, takes as input the key $K_t$ and an encoded message $m_t$ and a candidate tag $\tau'$ and outputs a bit. For any key $K_t$, message $m_t$ and internal state of $\mathcal{T}_{K_t}$, we require that $\mathcal{V}_{K_t}(m_t, \mathcal{T}_{K_t}(m_t)) = 1$. Security notions for MACs can be found in Appendix A.2.

### 3.3 Encode-then-Encrypt&MAC

With the above components defined, we are now ready to define SSH-CTR. Note that our version is significantly different from that considered in [5] because of the new features that we discussed in Section 2.1.

Our construction of SSH-CTR is an Encode-then-Encrypt&MAC construction with plaintext-dependent decryption. We define SSH-CTR = ($\mathcal{K}$-SSH-CTR, $\mathcal{E}$-SSH-CTR, $\mathcal{D}$-SSH-CTR) in Figure 3. This makes use of the encoding scheme $\mathcal{EC}$ described in Section 3.2, the encryption scheme CTR[$F$] and a message authentication scheme $\mathcal{MA}$, where the length of the MAC tag is `maclen`. It also makes use of a length checking algorithm len that we discuss below. Note that this construction is stateful. The encryption state arises from the counter mode state $ctr_e$ combined with the state $(st_e, SN_e)$ of the algorithm enc. The decryption state arises from the counter mode state $ctr_d$, the state $(st_d, SN_d)$ of the algorithm dec, and the ciphertext buffer `cbuff`. We will refer to the scheme SSH-CTR[$F$] whenever we wish to highlight the scheme's reliance on a particular function family $F$ in the encryption component.

The key generation algorithm $\mathcal{K}$-SSH-CTR selects keys for counter mode encryption and the MAC algorithm uniformly at random from the relevant key-spaces. This represents a significant abstraction from reality in our description of SSH-CTR, since in practice these keys and the initial counter value $ctr$ are derived in a pseudorandom manner from the keying material established during SSH's key exchange protocol. The decryption algorithm $\mathcal{D}$-SSH-CTR is considerably more complex than one might expect. This complexity is required to accurately model all the features of the SSH specification and the OpenSSH implementation. $\mathcal{D}$-SSH-CTR operates in 3 distinct stages.

In Stage 1, a sequence of ciphertext bytes $c$ of arbitrary length is received and appended to the ciphertext buffer `cbuff`.

In Stage 2 of $\mathcal{D}$-SSH-CTR, once sufficient bytes have arrived to process the first block of ciphertext, the packet length field is extracted, and length checking is performed by making a call

**Algorithm** $\mathcal{K}$-SSH-CTR($k$)
  $K_e \xleftarrow{r} \mathcal{K}_e(k)$
  $K_t \xleftarrow{r} \mathcal{K}_t(k)$
  $ctr \xleftarrow{r} \{0,1\}^l$
  **return** $K_e, K_t$

**Algorithm** $\mathcal{E}$-SSH-CTR$_{K_e,K_t}(m)$
  **if** $st_e = \perp$ **then**
    **return** $\perp$
  **end if**
  $(m_e, m_t) \leftarrow \mathsf{enc}(m)$
  **if** $m_e = \perp$ **then**
    $st_e \leftarrow \perp$
    **return** $\perp$
  **else**
    $c \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_e)$
    $\tau \leftarrow \mathcal{T}_{K_t}(m_t)$
    **return** $c \| \tau$
  **end if**

**Algorithm** $\mathsf{len}(m)$ ($|m| = L$)
  Parse $m$ as $\langle LF \rangle_4 \| R$
  **if** $LF \leq 5$ or $LF \geq 2^{18}$ **then**
    **return** $\perp_L$
  **else if** $LF + 4 \bmod L \neq 0$ **then**
    **return** $\perp_L$
  **else**
    **return** $LF$
  **end if**

**Algorithm** $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
  **if** $st_d = \perp$ **then**
    **return** $\perp$
  **end if**
  {*Stage 1*}
  $\mathtt{cbuff} \leftarrow \mathtt{cbuff} \| c$
  {*Stage 2*}
  **if** $m_e = \varepsilon$ and $|\mathtt{cbuff}| \geq L$ **then**
    Parse $\mathtt{cbuff}$ as $\tilde{c} \| A$ (where $|\tilde{c}| = L$)
    $m_e[1] \leftarrow \mathcal{D}\text{-CTR}_{K_e}(\tilde{c})$
    $LF \leftarrow \mathsf{len}(m_e[1])$
    **if** $LF = \perp_L$ **then**
      $st_d \leftarrow \perp$
      **return** $\perp_L$
    **else**
      $\mathtt{need} = 4 + LF + \mathtt{maclen}$
    **end if**
  **end if**
  {*Stage 3*}
  **if** $|\mathtt{cbuff}| \geq L$ **then**
    **if** $|\mathtt{cbuff}| \geq \mathtt{need}$ **then**
      Parse $\mathtt{cbuff}$ as $\bar{c}[1 \ldots n] \| \tau \| B$,
      where $|\bar{c}[1 \ldots n] \| \tau| = \mathtt{need}$,
      and $|\tau| = \mathtt{maclen}$
      $m_e[2 \ldots n] \leftarrow \mathcal{D}\text{-CTR}_{K_e}(\bar{c}[2 \ldots n])$
      $m_e \leftarrow m_e[1] \| m_e[2 \ldots n]$
      $m_t \leftarrow SN_d \| m_e$
      $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
      **if** $v = 0$ **then**
        $st_d \leftarrow \perp$
        **return** $\perp_A$
      **else**
        $m \leftarrow \mathsf{dec}(m_e)$
        $m_e \leftarrow \varepsilon$, $\mathtt{cbuff} \leftarrow B$
        **return** $m$
      **end if**
    **end if**
  **end if**

**Fig. 3.** SSH-CTR, SSH using counter mode encryption

to the function len. This accords with our discussion in Section 2.1. The function len is shown as part of Figure 3. It takes as input a single block of plaintext, and returns either the content of the length field (as an integer) or a failure symbol $\perp_L$. The exact details of length checking, and how to behave if length checking fails, is implementation-specific and not specified in the RFCs. Figure 3 shows the exact checks carried out by OpenSSH version 5.2 in counter mode; our subsequent analysis still holds so long as the algorithm at a minimum checks that the total number of encrypted bytes (i.e. excluding the MAC tag) indicated by the length field is a multiple of the block-size $L$, and fails if this is not the case. For further discussion, see Appendix A.1. Note that when length checking fails in OpenSSH version 5.2 in counter mode, an error message is sent and the SSH connection is torn down. We model this by outputting a length error $\perp_L$ and setting the state $st_d$ to $\perp$. Because the first action of $\mathcal{D}$-SSH-CTR is to simply return $\perp$ if $st_d$ is already equal to $\perp$, our description of SSH-CTR models the subsequent connection tear-down seen in OpenSSH. If the length checks pass, then $\mathcal{D}$-SSH-CTR proceeds to use the returned value of $LF$ to determine the value of need, which is the number of additional ciphertext bytes that are needed before the entire ciphertext (including MAC tag) is adjudged to have arrived. This makes the decryption

algorithm plaintext-dependent. However, it is not fully online, since no further output is produced by $\mathcal{D}$-SSH-CTR until the complete ciphertext has arrived and its MAC has been checked.

In Stage 3 of $\mathcal{D}$-SSH-CTR, ciphertext bytes that have been buffered in `cbuff` during Stage 1 are processed. Note that our model allows the recipient to receive more data than he expects; this data is denoted by $B$ in Stage 3. This data is assumed to be the start of the next ciphertext message and so we reinitialise `cbuff` with this data at the end of Stage 3. Once the buffer contains sufficient data (as determined by the variable `need`), the decryption algorithm uses counter mode to obtain the encoded plaintext $m_e$ and the message $m_t$ to be verified by the MAC algorithm (this consists of $m_e$ with the sequence number prepended). The MAC tag is then checked, and, if it verifies successfully, the encoded plaintext $m_e$ is passed to the `dec` algorithm (as defined in Figure 2). Notice that three types of error can arise during this stage: a failure of the MAC verification, resulting in output $\perp_A$, a failure of parsing during decoding, resulting in output $\perp_P$, or a wrap-around of the sequence number $SN_d$ during decoding, resulting in output $\perp$. When any of these errors arises, the state $st_d$ of the decryption algorithm is set as $\perp$. This state is checked at the start of every oracle query and if it equals $\perp$, then an error message $\perp$ is returned. In this way, our description of SSH-CTR models the subsequent connection tear-down seen in OpenSSH.

This description of SSH-CTR faithfully models the behaviour of OpenSSH in counter mode, in the sense of having buffered, plaintext-dependent decryption, and with errors arising at exactly the same points during decryption and based on the same failure conditions that are tested in OpenSSH. There are other ways in which to implement SSH and still be RFC-compliant. For example, the full decoding of the message, and hence parsing checks, could be performed before the MAC verification, as is the case in the construction of SSH-CTR given in [5]. However, it is not hard to derive distinguishing attacks against SSH-CTR when the OpenSSH parsing checks are carried out before MAC verification and the parsing and MAC failure errors are distinguishable (which they are not in [5]).

## 4 Security Models

### 4.1 Chosen Plaintext Security

We begin by extending the usual left-or-right (LOR) indistinguishability game for a CPA adversary from [4] to handle stateful encryption and leakage of length information. This extension is only needed at intermediate steps in our security analysis, while we are primarily interested in the security of the SSH BPP under chosen ciphertext attacks. For this reason, we content ourselves with chosen plaintext security definitions that are tied to the particular schemes SSH-CTR$[F]$ and CTR$^{\mathcal{EC}}[F]$ that we need to analyze.

In the usual LOR-CPA model the adversary is given access to a left-or-right encryption oracle $\mathcal{E}(\mathcal{LR}(\cdot,\cdot,b))$, where $b \in \{0,1\}$. This oracle takes as input two messages $m_0$ and $m_1$. If $b = 0$ it outputs the encryption of $m_0$ and if $b = 1$ it outputs the encryption of $m_1$. It is the adversary's challenge to determine the bit $b$. The advantage of such an adversary is defined in the usual way. Our extension of the LOR-CPA model makes it stateful and incorporates leakage of a length field. To achieve the former, we incorporate explicit sequence numbers in the model. To achieve the latter, we provide the adversary with access to a length revealing oracle $\mathcal{L}(\cdot)$ whose operation is specific to the particular scheme under study. For the schemes SSH-CTR$[F]$ and CTR$^{\mathcal{EC}}[F]$, the oracle takes as input a block $c$ which is treated as the first block of a new message; the oracle decrypts this block to retrieve the length field and performs the required length checking functions, and then outputs either the length field $LF$ or the symbol $\perp_L$ signifying an invalid length field. We require that $\mathcal{L}(\cdot)$ maintains its own view of any internal state of the underlying encryption scheme, according to the queries it receives. For the schemes we consider, this is done by increasing a counter value $ctr_l$ by a number that is determined by the length field, and increasing a sequence

number $SN_l$ by 1, each time the oracle is called; at the start of the security game, $ctr_l$ and $SN_l$ are set to the corresponding values held at the encryption oracle. The detailed operation of the length oracle associated with the schemes SSH-CTR[$F$] and CTR$^{\mathcal{EC}}$[$F$] can be found in Appendix A.2. We name our new model LOR-LLSF-CPA, where "LLSF" stands for "length leaking stateful".

In [5], decryption queries are defined to be either "in-sync" or "out-of-sync" with respect to the sequence number at the encryption oracle. We introduce a similar concept for length oracle queries in our next definition:

**Definition 1.** ***[LOR-LLSF-CPA]***
*Consider the stateful encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with an associated length oracle $\mathcal{L}(\cdot)$. Let $b \in \{0,1\}$ and $k \in \mathbb{N}$. Let $\mathcal{A}$ be an attacker that has access to the oracles $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ and $\mathcal{L}(\cdot)$ The game played is as follows:*

$$\begin{array}{l}\boldsymbol{Exp}^{\textit{lor-llsf-cpa-b}}_{\mathcal{E},\mathcal{A}}(k)\\ K \xleftarrow{r} \mathcal{K}(k)\\ b' \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{L}(\cdot)}\\ \textbf{\textit{return}}\ b'\end{array}$$

*For all queries $(m_0, m_1)$ to $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, we require that $|\mathsf{enc}(m_0)| = |\mathsf{enc}(m_1)|$. In this model the adversary has the possibility of making three different types of query to $\mathcal{L}$. Let $SN_e$ denote the sequence numbers at the encryption oracle and let $SN_l$ denote the sequence numbers at the length oracle.*

- *A query $c$ to $\mathcal{L}$ when the length oracle has sequence number $SN_l$ is said to be in-sync if $c$ is equal to the first block of ciphertext output by the encryption oracle when it had sequence number $SN_e = SN_l$.*
- *A query $c$ to $\mathcal{L}$ when the length oracle has sequence number $SN_l$ is said to be an out-of-sync current state query if $c$ is not equal to the first block of ciphertext output by the encryption oracle when it had sequence number $SN_e = SN_l$.*
- *A query to $\mathcal{L}$ when the length oracle has sequence number $SN_l$ is said to be an out-of-sync future state query if $SN_l > SN_e$, where $SN_e$ is the sequence number used by the encryption oracle when responding to its most recent query.*

*We require that the response to any further length oracle queries following the first out-of-sync query is $\perp$.*

*The attacker wins when $b' = b$, and its advantage is defined to be:*

$$\boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{\mathcal{SE},\mathcal{A}}(k) = \Pr[\boldsymbol{Exp}^{\textit{lor-llsf-cpa-1}}_{\mathcal{SE},\mathcal{A}}(k) = 1] - \Pr[\boldsymbol{Exp}^{\textit{lor-llsf-cpa-0}}_{\mathcal{SE},\mathcal{A}}(k) = 1].$$

*The advantage function of the scheme is defined to be*

$$\boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{\mathcal{SE}}(k, t, q_e, \mu_e, q_l) = \max_{\mathcal{A}}\{\boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{\mathcal{SE},\mathcal{A}}(k)\}$$

*for any integers $t, q_e, \mu_e, q_l$. The maximum is over all adversaries $\mathcal{A}$ with time complexity $t$, making at most $q_e$ queries to the encryption oracle, totalling at most $\mu_e$ bits in each of the left and right inputs, and $q_l$ queries to the length revealing oracle.*

### 4.2 Chosen Ciphertext Security

Now we consider chosen ciphertext attackers[1]. We introduce a new security notion for left-or-right indistinguishability against chosen-ciphertext attackers for buffered, stateful decryption (LOR-BSF-CCA). In this model, which extends the IND-SFCCA model of [5], the adversary is given

[1] Note that an online encryption scheme cannot be CCA secure [8]. Despite the underlying encryption scheme we consider being online, the combined encryption scheme we consider is not online. We therefore do not need to be concerned about the trivial attack raised in [8].

access to an encryption oracle and to a *buffered* decryption oracle. The model applies for any encryption scheme in which the decryption oracle maintains a buffer of as-yet-unprocessed ciphertext bytes `cbuff` and in which encryption and decryption states include sequence numbers which are incremented after each successful operation. For reasons explained in Section 2.1, we need to limit the attacker's queries to the encryption oracle to pairs of messages $(m_0, m_1)$ having the same length when encoded.

**Definition 2.** ***[LOR-BSF-CCA]***
*Consider the symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with buffered, stateful decryption. Let $b \in \{0,1\}$ and $k \in \mathbb{N}$. Let $\mathcal{A}$ be an attacker that has access to the oracles $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ and $\mathcal{D}_K(\cdot)$. The game played is as follows:*

$$\begin{array}{l}\boldsymbol{Exp}^{\textit{lor-bsf-cca-b}}_{\mathcal{SE},\mathcal{A}}(k)\\ K \xleftarrow{r} \mathcal{K}(k)\\ b' \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{D}_K(\cdot)}(k)\\ \textbf{\textit{return}}\ b'\end{array}$$

*We require that for all queries $(m_0, m_1)$ to $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, $|\mathsf{enc}(m_0)| = |\mathsf{enc}(m_1)|$. In this model the adversary has the possibility of making three different types of decryption query. Let $SN_e$ denote the sequence numbers at the encryption oracle and let $SN_d$ denote the sequence numbers at the decryption oracle. Recall that, since the adversary can deliver ciphertexts in a byte-wise fashion to the decryption oracle, the same value of $SN_d$ may be involved in processing a sequence of ciphertext queries.*

- *The sequence of decryption queries corresponding to the sequence number $SN_d$ is said to be in-sync if, after input of the final query in the sequence, the ciphertext buffer* `cbuff` *has as a prefix the output from the encryption oracle for sequence number $SN_e = SN_d$. The response from an in-sync query is not returned to the adversary.*
- *The sequence of decryption queries corresponding to the sequence number $SN_d$ is said to be an out-of-sync current state query if, after input of the final query in the sequence, the ciphertext buffer* `cbuff` *does not have the output from the encryption oracle for sequence number $SN_e = SN_d$ as a prefix.*
- *The sequence of decryption queries corresponding to the sequence number $SN_d$ is said to be an out-of-sync future state query if $SN_d > SN_e$, where $SN_e$ is the sequence number used by the encryption oracle when responding to its most recent query.*

*The response to any further decryption queries following an out-of-sync query is the $\perp$ symbol.*

*The attacker wins when $b' = b$, and its advantage is defined to be:*

$$\boldsymbol{Adv}^{\textit{lor-bsf-cca}}_{\mathcal{SE},\mathcal{A}}(k) = \Pr[\boldsymbol{Exp}^{\textit{lor-bsf-cca-1}}_{\mathcal{SE},\mathcal{A}}(k) = 1] - \Pr[\boldsymbol{Exp}^{\textit{lor-bsf-cca-0}}_{\mathcal{SE},\mathcal{A}}(k) = 1].$$

*The advantage function of the scheme is defined to be*

$$\boldsymbol{Adv}^{\textit{lor-bsf-cca}}_{\mathcal{SE}}(k, t, q_e, \mu_e, q_d, \mu_d) = \max_{\mathcal{A}}\{\boldsymbol{Adv}^{\textit{lor-bsf-cca}}_{\mathcal{SE},\mathcal{A}}(k)\}$$

*for any integers $t, q_e, \mu_e, q_d, \mu_d$. The maximum is over all adversaries $\mathcal{A}$ with time complexity $t$, making at most $q_e$ queries to the encryption oracle, totalling at most $\mu_e$ bits in each of the left and right inputs, and at most $q_d$ series of queries to the decryption oracle, totalling at most $\mu_d$ bits.*

In the model above, the response from an in-sync decryption query is not returned to the adversary. This is required in order to prevent the obvious and trivial attack in which the adversary simply queries the decryption oracle with the output from the encryption oracle. We include in-sync

decryption queries in order to permit the adversary to observe the system's behaviour in encrypting messages of its choice and to let the adversary advance the sequence numbers maintained at the encryption and decryption oracles to values of its choice. We make the restriction that only one out-of-sync query is allowed for the same reason that this restriction is made in [5]: if the first out-of-sync query does not decrypt successfully, the decryption oracle enters a halting state anyway, while if it does, then our security analysis will show that the adversary has broken the strong unforgeability of the MAC scheme. Our security model and analysis can be extended to handle multiple out-of-sync decryption queries.

The specific decryption oracle that we consider when analyzing the security of SSH-CTR operates exactly as the decryption algorithm $\mathcal{D}$-SSH-CTR in Section 3.3: the oracle takes as input an arbitrary number of bytes which is then added to `cbuff`; the decryption process uses the first plaintext block to determine how many bytes of ciphertext are needed to complete the packet; and the decryption process involves length checking, MAC checking, and decoding, with each of these steps potentially outputting a distinct error message. Also note that for SSH-CTR, the decryption oracle acts as a "bomb" oracle: when an error of any type occurs this oracle simply outputs $\perp$ in response to any further query. This models an attempt by the decrypting party to initiate an SSH connection tear-down. However, note that our model for SSH-CTR has separate states for encryption and decryption, so that the encryption oracle is not "lost" if the decryption oracle is. This allows us to model an adversary that outputs the relevant error messages. This description of SSH-CTR in the context of the LOR-BSF-CCA model is sufficiently rich to give the attacker all the capabilities exploited in the attacks of Albrecht *et al.* [1]. Thus, if we can prove SSH-CTR to be secure in the LOR-BSF-CCA sense, then attacks of the kind developed in [1] will be prevented.

### 4.3 Integrity of Ciphertexts

We next extend the INT-SFCTXT model from [5] to include buffered decryption. We call our new model "integrity of ciphertexts for buffered, stateful decryption" or INT-BSF-CTXT. The model again applies for any encryption scheme in which the decryption oracle maintains a buffer of as-yet-unprocessed ciphertext bytes `cbuff` and in which encryption and decryption states include sequence numbers which are incremented after each successful operation.

**Definition 3.** ***[INT-BSF-CTXT]***
*Consider the symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with buffered, stateful decryption, and let $k \in \mathbb{N}$. Let $\mathcal{A}$ be an attacker that has access to the oracles $\mathcal{E}_K(\cdot)$ and $\mathcal{D}_K(\cdot)$. In this game the adversary again has the possibility of making three different types of decryption query: in-sync, out-of-sync current state and out-of-sync future state. These are defined exactly as in the LOR-BSF-CCA security game above. The response to any further decryption queries following an out-of-sync query is the $\perp$ symbol. The game played is as follows:*

$\boldsymbol{Exp}^{\textit{int-bsf-ctxt}}_{\mathcal{SE},\mathcal{A}}(k)$
$K \xleftarrow{r} \mathcal{K}(k)$
***if*** $\mathcal{A}^{\mathcal{E}_K(\cdot),\mathcal{D}_K(\cdot)}(k)$ *makes an out-of-sync sequence of queries to the decryption oracle* $\mathcal{D}_K(\cdot)$ *such that:*
– *there is an output from the decryption oracle; and*
– *the output is not a member of the set* $\{\perp_L, \perp_A, \perp_P, \perp\}$.
*then* ***return*** 1
*else* ***return*** 0

*The attacker's advantage is defined to be:*

$$\boldsymbol{Adv}^{\textit{int-bsf-ctxt}}_{\mathcal{SE},\mathcal{A}}(k) = \Pr[\boldsymbol{Exp}^{\textit{int-bsf-ctxt}}_{\mathcal{SE},\mathcal{A}}(k) = 1].$$

*The advantage function of the scheme is defined to be*

$$\boldsymbol{Adv}^{\textit{int-bsf-ctxt}}_{\mathcal{SE}}(k,t,q_e,\mu_e,q_d,\mu_d) = \max_{\mathcal{A}}\{\boldsymbol{Adv}^{\textit{int-bsf-ctxt}}_{\mathcal{SE},\mathcal{A}}(k)\}$$

*for any integers $t, q_e, \mu_e, q_d, \mu_d$. The maximum is over all $\mathcal{A}$ with time complexity $t$, each making at most $q_e$ queries to the encryption oracle, totalling at most $\mu_e$ bits, and at most $q_d$ series of queries to the decryption oracle, totalling at most $\mu_d$ bits.*

Again, the specific decryption oracle that we consider when analyzing the security of SSH-CTR operates exactly as the decryption algorithm $\mathcal{D}$-SSH-CTR in Section 3.3.

### 4.4 Security of Message Authentication Schemes

Finally, we define two security notions for MACs. We will use the LOR-DCPA notion from [5], for distinct plaintext privacy of message authentication schemes. We will also use the standard SUF-CMA model for strong unforgeability of MACs. The formal definitions for these notions can be found in Appendix A.2.

## 5 Security Analysis

We will now present our main result, Theorem 1. This theorem provides a concrete security guarantee for the scheme SSH-CTR[$F$] in terms of security properties of the pseudorandom function family $F$ and MAC scheme $\mathcal{MA}$ used in its construction. The structure of our proof follows that in [5], but with significant modifications being needed to handle the new features of our security model and adversary. Our proof is valid no matter what length checks are performed by the encoding scheme, so long as the minimal length check described previously is included. Our proof is also valid (and in fact can be tightened slightly) if the random padding bytes in the encoding scheme are replaced by fixed bytes. It is also valid no matter what specific parsing checks are carried out, provided that the encoding scheme is correct. With the exception of our main result, the proofs can all be found in Appendix B.

**Theorem 1.** *Let SSH-CTR[$F$] be the combined encryption scheme for the encoding scheme $\mathcal{EC}$, counter mode encryption CTR[$F$] and a message authentication scheme $\mathcal{MA}$. Then SSH-CTR[$F$] is LOR-BSF-CCA secure if $F$ is a pseudorandom function family, if $\mathcal{T}$ (the tagging algorithm from $\mathcal{MA}$) is a pseudorandom function family, and if $\mathcal{MA}$ is SUF-CMA secure. Concretely, for $q_e, q_d \leq 2^{32}$, $\mu_e \leq 8L2^l - 8q_e(8+L)$ and any $t, k, \mu_d$, we have:*

$$\begin{aligned}&\boldsymbol{Adv}^{\textit{lor-bsf-cca}}_{\textit{SSH-CTR}[F]}(k,t,q_e,\mu_e,q_d,\mu_d)\\ &\quad\leq 2\boldsymbol{Adv}^{\textit{suf-cma}}_{\mathcal{MA}}(k,t,q_t,\mu_t,q_v,\mu_v) + 2\boldsymbol{Adv}^{\textit{prf}}_{F}(k,t',q_F) + 4\boldsymbol{Adv}^{\textit{prf}}_{\mathcal{T}}(k,t'',q_t)\end{aligned}$$

*where $q_t = q_e$, $\mu_t \leq \mu_e + 8(L+12)q_e$, $q_v = q_d$, $\mu_v \leq \mu_d + 32q_d$, $q_F \leq q_l + \mu_e/8L + q_e(1+8/L)$, $t' = O(t)$ and $t'' = O(t)$.*

**Proof of Theorem 1:** This follows from Theorem 2 and Lemmas 1, 2, 3, 4 and 5. □

The following is an extension of a result of Bellare and Namprempre [7]; here we consider buffered, stateful decryption and include in our model potential errors arising from length checking, MAC failures and parsing failures.

**Theorem 2.** *Let SSH-CTR[$F$] be the combined encryption scheme for the encoding scheme $\mathcal{EC}$, counter mode encryption CTR[$F$] and a message authentication scheme $\mathcal{MA}$. Then SSH-CTR[$F$]*

*is LOR-BSF-CCA secure if it is both INT-BSF-CTXT and LOR-LLSF-CPA secure. Concretely, for any $k, t, q_e, \mu_e, q_d, \mu_d$, we have:*

$$\begin{aligned}&\boldsymbol{Adv}^{\textit{lor-bsf-cca}}_{\textit{SSH-CTR}[F]}(k, t, q_e, \mu_e, q_d, \mu_d)\\ &\quad \leq 2\boldsymbol{Adv}^{\textit{int-bsf-ctxt}}_{\textit{SSH-CTR}[F]}(k, t, q_e, \mu_e, q_d, \mu_d) + \boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{\textit{SSH-CTR}[F]}(k, t, q_e, \mu_e, q_l)\end{aligned}$$

*where $q_l = q_d$.*

**Lemma 1.** *Let SSH-CTR[F] be the combined encryption scheme for the encoding scheme $\mathcal{EC}$, counter mode encryption CTR[F] and a message authentication scheme $\mathcal{MA}$. Then SSH-CTR[F] is INT-BSF-CTXT secure if $\mathcal{MA}$ is SUF-CMA secure. More concretely, for $q_e, q_d \leq 2^{32}$ and any $k, t, \mu_e, \mu_d$, we have:*

$$\boldsymbol{Adv}^{\textit{int-bsf-ctxt}}_{\textit{SSH-CTR}[F]}(k, t, q_e, \mu_e, q_d, \mu_d) \leq \boldsymbol{Adv}^{\textit{suf-cma}}_{\mathcal{MA}}(k, t, q_t, \mu_t, q_v, \mu_v)$$

*where $q_t = q_e$, $\mu_t \leq \mu_e + 8(L + 12)q_e$, $q_v = q_d$, and $\mu_v \leq \mu_d + 32q_d$.*

Now recall the definition of the scheme $\mathrm{CTR}^{\mathcal{EC}}[F]$ that combines the encoding scheme and counter mode encryption from Section 3.2 (for the details of the scheme, see Appendix A.1).

**Lemma 2.** *Let SSH-CTR[F] be the combined encryption scheme for the encoding scheme $\mathcal{EC}$, counter mode encryption CTR[F] and a message authentication scheme $\mathcal{MA}$. Then SSH-CTR[F] is LOR-LLSF-CPA secure if $CTR^{\mathcal{EC}}[F]$ is LOR-LLSF-CPA secure and $\mathcal{MA}$ is LOR-DCPA secure. More concretely, for $q_e, q_l \leq 2^{32}$ and any $k, t, \mu_e$, we have:*

$$\begin{aligned}&\boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{\textit{SSH-CTR}[F]}(k, t, q_e, \mu_e, q_l)\\ &\quad \leq \boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{CTR^{\mathcal{EC}}[F]}(k, t', q_e, \mu_e, q_l) + 2\boldsymbol{Adv}^{\textit{lor-dcpa}}_{\mathcal{MA}}(k, t'', q_t, \mu_t)\end{aligned}$$

*where $q_t = q_e$, $t' = O(t)$, $t'' = O(t)$, and $\mu_t \leq \mu_e + 16(L + 12)q_e$.*

**Lemma 3.** *Suppose $F$ is a pseudorandom function family with input length $l$ bits and output length $L$ bytes. Let $R = Rand^{l \to L}$ be the set of all functions mapping $l$-bit strings to $L$-byte strings. Then for any $k, t, q_e, \mu_e, q_l$, we have:*

$$\boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{CTR^{\mathcal{EC}}[F]}(k, t, q_e, \mu_e, q_l) \leq 2\boldsymbol{Adv}^{\textit{prf}}_{F}(k, t', q_F) + \boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{CTR^{\mathcal{EC}}[R]}(k, t, q_e, \mu_e, q_l)$$

*where $q_F \leq q_l + \mu_e/8L + q_e(40 + 8(3 + L))/8L$ and $t' = O(t)$.*

**Lemma 4.** *For any $k, t, q_l$, $q_e$ and $\mu_e \leq 8L2^l - 8q_e(8 + L)$ we have:*

$$\boldsymbol{Adv}^{\textit{lor-llsf-cpa}}_{CTR^{\mathcal{EC}}[R]}(k, t, q_e, \mu_e, q_l) = 0.$$

**Lemma 5.** *Let $\mathcal{MA}$ be a message authentication scheme. Then $\mathcal{MA}$ is LOR-DCPA secure if $\mathcal{T}$, the tagging algorithm from $\mathcal{MA}$, is a pseudorandom function family. More concretely, for any $k, t$ and $q_t$, we have:*

$$\boldsymbol{Adv}^{\textit{lor-dcpa}}_{\mathcal{MA}}(k, t, q_t, \mu_t) \leq 2\boldsymbol{Adv}^{\textit{prf}}_{\mathcal{T}}(k, t', q_t)$$

*where $t' = O(t)$.*

## 6 Conclusion

We have extended the security model of Bellare *et al.* [5] to develop a model suited to analyzing the SSH BPP. We gave a description of SSH-CTR that is closely linked to the specification of SSH in the RFCs and the OpenSSH implementation of SSH. We then proved the security of SSH-CTR in the extended model. Our approach is sufficiently powerful to incorporate the attacks of Albrecht *et al.* [1]. This helps to close the gap that exists between the formal security analysis of SSH and the way in which SSH should be (and is in practice) implemented.

Our approach can be seen as an attempt to expand the scope of provable security to incorporate the fine details of cryptographic implementations. We grant the attacker a much wider and more realistic set of ways of interacting with the SSH protocol than in the previous analysis of [5]. We believe that our approach captures more of the cryptographically relevant features of the SSH BPP, including plaintext-dependent, byte-wise decryption and detailed modeling of the errors that can arise during cryptographic processing in the SSH BPP.

One drawback of our approach is that, while we have tried to be as general as possible in our modeling of SSH, our security results are now specific to one implementation of the SSH RFCs, namely OpenSSH. This seems to us to be an inevitable consequence of making the model and the security analysis more realistic. We leave it as an interesting open question as to whether similar analyses can be carried out for other network protocols. A prime example would be SSL/TLS, where the current security analysis of [15] is not stateful and does not model errors or padding, whilst attacks exploiting such features of real SSL/TLS implementations were reported in [9]. Our general approach appears to open up a rich new seam of research within provable security.

## References


1. M.R. Albrecht, K.G. Paterson and G.J. Watson. Plaintext recovery attacks against SSH. In *IEEE Symposium on Security and Privacy*, pages 16–26. IEEE Computer Society, 2009.
2. G.V. Bard. A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL. In M. Malek, E. Fernández-Medina and J. Hernando, eds., *SECRYPT*, pages 99–109. INSTICC Press, 2006.
3. G.V. Bard. Blockwise-adaptive chosen-plaintext attack and online modes of encryption. In S.D. Galbraith, ed., *IMA Int. Conf.*, volume 4887 of *Lecture Notes in Computer Science*, pages 129–151. Springer, 2007.
4. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 394–403. IEEE, 1997.
5. 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 Systems Security*, 7(2):206–241, 2004.
6. M. Bellare, T. Kohno, and C. Namprempre. The Secure Shell (SSH) Transport Layer Encryption Modes. RFC 4344, January 2006. `http://www.ietf.org/rfc/rfc4344.txt`.
7. M. Bellare and C Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, ed., *Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
8. A. Boldyreva and N. Taesombut. Online encryption schemes: New security notions and constructions. In T. Okamoto, ed., *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2004.
9. B. Canvel, A.P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TLS channel. In D. Boneh, ed., *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 583–599, 2003.
10. CPNI Vulnerability Advisory. Plaintext recovery attack against SSH, 14/11/2008 (revised 17/11/2008). `http://www.cpni.gov.uk/Docs/Vulnerability_Advisory_SSH.txt`
11. P.-A. Fouque, A. Joux, G. Martinet and F. Valette. Authenticated on-line encryption. In M. Matsui and R.J. Zuccherato, eds., *SAC*, volume 3006 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2003.
12. P.-A. Fouque, A. Joux and G. Poupard. Blockwise adversarial model for on-line ciphers and symmetric encryption schemes. In H. Handschuh and M.A. Hasan, eds., *SAC*, volume 3357 of *Lecture Notes in Computer Science*, pages 212–226. Springer, 2004.
13. P.-A. Fouque, G. Martinet and G. Poupard. Practical symmetric on-line encryption. In T. Johansson, ed., *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 362–375. Springer, 2003.

14. A. Joux, G. Martinet and F. Valette. Blockwise-adaptive attackers: Revisiting the (in)security of some provably secure encryption models: CBC, GEM, IACBC. In M. Yung, ed., *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2002.
15. H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, ed., *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310-331, 2001.
16. SSH usage profiling, `http://www.openssh.org/usage/index.html`.
17. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006. `http://www.ietf.org/rfc/rfc4251.txt`.
18. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252, January 2006. `http://www.ietf.org/rfc/rfc4252.txt`.
19. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, January 2006. `http://www.ietf.org/rfc/rfc4253.txt`.
20. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254, January 2006. `http://www.ietf.org/rfc/rfc4254.txt`.

# A Further Definitions

## A.1 Building Block Definitions

**Length Checking for OpenSSH:** The definition of algorithm `len` in Figure 3 showed the specific length checks used in OpenSSH version 5.2 (and more recent versions) in counter mode as an example of the kind of checks that may be performed. As can be seen from that figure, OpenSSH version 5.2 in counter mode performs two length checks: a packet length check which verifies that the length field is at least 5 and less than $2^{18}$ and a block length check which verifies that the amount of data that is expected to be received is a multiple of the block size. If either of these checks fails the session is terminated in exactly the same way, to ensure uniform reporting of errors arising from length checks. This uniformity in checking was adopted as an initial response to the attacks of Albrecht *et al.* [1]; the previous versions of OpenSSH allowed the different error conditions to be distinguished. OpenSSH version 5.2 in CBC mode uses additional countermeasures in order to make length errors and MAC verification errors indistinguishable, thus preventing the attacks of [1]. This means that the length checking carried out by OpenSSH version 5.2 is different for CBC mode and counter mode. Here, we consider only counter mode.

**Pseudorandom Function Family:** We give a definition for a pseudorandom function family. Note that our definition requires all functions to have $l$-bit inputs and $L$-byte outputs. This is non-standard, but fits with our use of an $l$-bit counter for counter mode and an $L$-byte block-size.

**Definition 4.** ***[Pseudorandom Function (prf) Family]*** *Let $F = \{F_K : K \in \mathcal{K}\}$ and $F_K : \{0,1\}^l \to \{0,1\}^{8L}$. Here $l$, $L$, the function family $F$ and its key-space $\mathcal{K}$ depend on a security parameter $k$. Let $Rand^{l\to L}$ be the set of all functions mapping $l$-bit strings to $L$-byte strings. Let $b \in \{0,1\}$. Let $D$ be a distinguisher with access to an oracle for the function $f_b(\cdot)$ as defined in the following experiment:*

$\boldsymbol{Exp}^{prf\text{-}b}_{F,D}(k)$
$f_0 \xleftarrow{r} Rand^{l\to L}$; $K \xleftarrow{r} \mathcal{K}$, $f_1 \leftarrow F_K$
$b' \leftarrow D^{f_b(\cdot)}$
***return*** $b'$

*The advantage of $D$ is defined as:*

$$\boldsymbol{Adv}^{prf}_{F,D}(k) = \Pr[\boldsymbol{Exp}^{prf\text{-}1}_{F,D}(k) = 1] - \Pr[\boldsymbol{Exp}^{prf\text{-}0}_{F,D}(k) = 1].$$

*The advantage function of the function family $F$ is defined as follows. For any integers $k, t, q_F$*

$$\boldsymbol{Adv}^{prf}_{F}(k,t,q_F) = \max_D\{\boldsymbol{Adv}^{prf}_{F,D}(k)\}$$

*where the maximum is over all $D$ with time complexity $t$, each making at most $q_F$ queries to the oracle.*

**Algorithm** $\mathcal{K}$-CTR$_K(k)$
$K_e \xleftarrow{r} \mathcal{K}_e(k)$
$ctr \xleftarrow{r} \{0,1\}^l$
**return** $K_e, ctr$

**Algorithm** $\mathcal{E}$-CTR$_K(m)$
Parse $m$ as $m[1]m[2]\ldots m[n]$,
where $|m[i]| = L, \forall i$
**for** $i = 1$ to $n$ **do**
$c[i] = F_K(ctr_e + i) \oplus m_i$
**end for**
$ctr_e \leftarrow ctr_e + n$
**return** $c[1]\ldots c[n]$

**Algorithm** $\mathcal{D}$-CTR$_K(c)$
Parse $c$ as $c[1]c[2]\ldots c[n]$,
where $|c[i]| = L, \forall i$
**for** $i = 1$ to $n$ **do**
$m[i] = F_K(ctr_d + i) \oplus c_i$
**end for**
$ctr_d \leftarrow ctr_d + n$
**return** $m[1]\ldots m[n]$

**Fig. 4.** Counter mode encryption

**Counter Mode Encryption:** We define the algorithms associated with counter mode encryption, CTR$[F]$ in Figure 4. Here $F$ is assumed to be a function family with key-space $\mathcal{K}_e(k)$, $l$-bit inputs and $L$-byte outputs. The key generation algorithm $\mathcal{K}$-CTR$_K^{\mathcal{EC}}(k)$ generates the encryption key $K_e$ and initial counter value $ctr$; the counter values $ctr_e$ and $ctr_d$ are (implicitly) assumed to be initialised to this value.

**Counter Mode Encryption and Encoding Scheme Combined:** We give the algorithms for the scheme CTR$^{\mathcal{EC}}[F]$ in Figure 5. This is essentially the same as the full scheme SSH-CTR$[F]$ with the omission of MAC tags. The key generation algorithm $\mathcal{K}$-CTR$_K^{\mathcal{EC}}(k)$ generates the encryption $K_e$ and initial counter value $ctr$ for counter mode encryption.

### A.2 Security Definitions

**Distinct Chosen Plaintext Attacks:**

**Definition 5.** ***[LOR-DCPA]***
*Consider the message authentication scheme $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$. Let $b \in \{0,1\}$ and $k \in \mathbb{N}$. Let $\mathcal{A}$ be an attacker that has access to the oracle $\mathcal{T}_{K_t}(\mathcal{LR}(\cdot,\cdot,b))$ The game played is as follows:*

$$\begin{array}{l} \boldsymbol{Exp}^{\textit{lor-dcpa-b}}_{\mathcal{MA},\mathcal{A}}(k) \\ K_t \xleftarrow{r} \mathcal{K}_t(k) \\ b' \leftarrow \mathcal{A}^{\mathcal{T}_{K_t}(\mathcal{LR}(\cdot,\cdot,b))} \\ \boldsymbol{return}\ b' \end{array}$$

*Here, it is required that all "left" messages in $\mathcal{A}$'s queries be distinct and that all "right" messages in $\mathcal{A}$'s queries be distinct. The attacker wins when $b' = b$, and its advantage is defined to be:*

$$\boldsymbol{Adv}^{\textit{lor-dcpa}}_{\mathcal{MA},\mathcal{A}}(k) = \Pr[\boldsymbol{Exp}^{\textit{lor-dcpa-1}}_{\mathcal{MA},\mathcal{A}}(k) = 1] - \Pr[\boldsymbol{Exp}^{\textit{lor-dcpa-0}}_{\mathcal{MA},\mathcal{A}}(k) = 1].$$

*The advantage function of the scheme is defined to be*

$$\boldsymbol{Adv}^{\textit{lor-dcpa}}_{\mathcal{MA}}(k, t, q_t, \mu_t) = \max_{\mathcal{A}}\{\boldsymbol{Adv}^{\textit{lor-dcpa}}_{\mathcal{MA},\mathcal{A}}(k)\}$$

*for any integers $t, q_t, \mu_t$. The maximum is over all adversaries $\mathcal{A}$ with time complexity $t$, making at most $q_t$ queries to the tag oracle, totalling at most $\mu_t$ bits.*

**Strong Unforgeability:**

**Definition 6.** ***[SUF-CMA]***
*Consider the message authentication scheme $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$. Let $b \in \{0,1\}$ and $k \in \mathbb{N}$. Let $\mathcal{F}$ be a forger that has access to the oracle $\mathcal{T}_{K_t}(\cdot)$ The game played is as follows:*

$\boldsymbol{Exp}^{\textit{suf-cma}}_{\mathcal{MA},\mathcal{F}}(k)$
$K_t \xleftarrow{r} \mathcal{K}_t(k)$
***if*** $\mathcal{F}^{\mathcal{T}_K(\cdot),\mathcal{V}_K(\cdot,\cdot)}$ *makes a query* $(m,\tau)$ *to* $\mathcal{V}_K(\cdot,\cdot)$ *such that*
- $\mathcal{V}_K(m,\tau)$ *returns* 1*, and*
- $\tau$ *was a never a response from* $\mathcal{T}_K(\cdot)$ *for a query* $m$*,*
*then* ***return*** *1 else* ***return*** *0*

*The forger's advantage is defined to be:*

$$\boldsymbol{Adv}^{\textit{suf-cma}}_{\mathcal{MA},\mathcal{F}}(k) = \Pr[\boldsymbol{Exp}^{\textit{suf-cma}}_{\mathcal{MA},\mathcal{F}}(k) = 1].$$

*The advantage function of the scheme is defined to be*

$$\boldsymbol{Adv}^{\textit{suf-cma}}_{\mathcal{MA}}(k,t,q_t,\mu_t,q_v,\mu_v) = \max_{\mathcal{F}}\{\boldsymbol{Adv}^{\textit{suf-cma}}_{\mathcal{MA},\mathcal{F}}(k)\}$$

*for any integers* $t, q_t, \mu_t, q_v, \mu_v$*. The maximum is over all forgers* $\mathcal{F}$ *with time complexity* $t$*, making at most* $q_t$ *queries to the tag oracle, totalling at most* $\mu_t$ *bits and making at most* $q_v$ *queries to the verification oracle, totalling at most* $\mu_v$ *bits.*

**Length Oracle for Counter Mode Encryption:** In Figure 6 we define the length oracle associated to the schemes SSH-CTR$[F]$ and CTR$^{\mathcal{EC}}[F]$. This oracle maintains its own state, defined

**Algorithm** $\mathcal{K}$-CTR$^{\mathcal{EC}}_K(k)$
$K_e \xleftarrow{r} \mathcal{K}_e(k)$
$ctr \xleftarrow{r} \{0,1\}^l$
**return** $K_e, ctr$

**Algorithm** $\mathcal{E}$-CTR$^{\mathcal{EC}}_{K_e}(m)$
**if** $st_e = \perp$ **then**
**return** $\perp$
**end if**
$(m_e, m_t) \leftarrow \mathsf{enc}(m)$
**if** $m_e = \perp$ **then**
$st_e \leftarrow \perp$
**return** $\perp$
**else**
$c \leftarrow \mathcal{E}$-CTR$_{K_e}(m_e)$
**return** $c$
**end if**

**Algorithm** $\mathsf{len}(m)$ $(|m| = L)$
Parse $m$ as $\langle LF\rangle_4 \| R$
**if** $LF \leq 5$ or $LF \geq 2^{18}$ **then**
**return** $\perp_L$
**else if** $LF + 4 \bmod L \neq 0$ **then**
**return** $\perp_L$
**else**
**return** $LF$
**end if**

**Algorithm** $\mathcal{D}$-CTR$^{\mathcal{EC}}_{K_e}(c)$
**if** $st_d = \perp$ **then**
**return** $\perp$
**end if**
{*Stage 1*}
$\mathtt{cbuff} \leftarrow \mathtt{cbuff} \| c$
{*Stage 2*}
**if** $m_e = \varepsilon$ and $|\mathtt{cbuff}| \geq L$ **then**
Parse $\mathtt{cbuff}$ as $\bar{c}\|A$, $(|\bar{c}| = L)$
$m_e[1] \leftarrow \mathcal{D}$-CTR$_{K_e}(\bar{c})$
$LF \leftarrow \mathsf{len}(m_e[1])$
**if** $LF = \perp_L$ **then**
$st_d \leftarrow \perp$
**return** $\perp_L$
**end if**
**end if**
$\mathtt{need} = 4 + LF$
{*Stage 3*}
**if** $|\mathtt{cbuff}| \geq L$ **then**
**if** $|\mathtt{cbuff}| \geq \mathtt{need}$ **then**
Parse $\mathtt{cbuff}$ as $\bar{c}[1\ldots n]\|B$,
where $|\bar{c}[1\ldots n]| = \mathtt{need}$
$m_e[2\ldots n] \leftarrow \mathcal{D}$-CTR$_{K_e}(\bar{c}[2\ldots n])$
$m_e \leftarrow m_e[1]\|m_e[2\ldots n]$
$m \leftarrow \mathsf{dec}(m_e)$
$m_e \leftarrow \varepsilon, \mathtt{cbuff} \leftarrow B$
**return** $m$
**end if**
**end if**

**Fig. 5.** $CTR^{\mathcal{EC}}[F]$: Counter mode and encoding scheme combined

```
Algorithm L(c) (|c| = L)
  if st_l = ⊥ then
    return ⊥
  end if
  m ← F_{K_e}(ctr_l) ⊕ c
  LF ← len(m)
  if LF = ⊥_L then
    st_l ← ⊥
    return ⊥_L
  else
    ctr_l ← ctr_l + ((LF + 4)/L)
    SN_l ← SN_l + 1
    return LF
  end if
```

**Fig. 6.** Length oracle for counter mode encryption

by a triple $(st_l, SN_l, ctr_l)$, initialised to $(\varepsilon, 0, ctr_e)$ where $ctr_e$ denotes the counter value held by the encryption oracle at the start of the LOR-LLSF-CPA security game. In order to update the counter value, the algorithm uses the length field to determine how many more blocks should be in the ciphertext, adding $(LF + 4)/L$ to the counter. Notice how we rely on the minimal length checking carried out by algorithm $\mathsf{len}$ here.

## B Proofs

**Proof of Theorem 2:** Let $\mathcal{A}$ be an adversary attacking SSH-CTR[$F$] in the LOR-BSF-CCA sense. We use this adversary to construct two new adversaries $\mathcal{B}, \mathcal{I}$ using their oracles to provide simulations of $\mathcal{A}$'s oracles, such that:

- $\mathcal{B}$ attacks SSH-CTR[$F$] in the LOR-LLSF-CPA sense.
- $\mathcal{I}$ attacks SSH-CTR[$F$] in the INT-BSF-CTXT sense.

The constructions of these adversaries can be found in Figure 7. In $\mathcal{B}$'s construction, $\mathcal{B}$ runs $\mathcal{A}$ using its own encryption oracle to provide simulations of $\mathcal{A}$'s encryption oracle and using its length oracle to provide simulations of $\mathcal{A}$'s decryption oracle, responding with $\perp_A$ if the decryption query was out-of-sync. Note that $\mathcal{B}$ makes at most one out-of-sync query to its length oracle, so is a valid LOR-LLSF-CPA adversary. In $\mathcal{I}$'s construction, $\mathcal{I}$ runs $\mathcal{A}$ using its own oracles to provide simulations of $\mathcal{A}$'s oracles.

Now let us consider the event $E$ that $\mathcal{A}$ makes an out-of-sync sequence of decryption oracle queries for which the output of the decryption oracle is not an element of $\{\perp_A, \perp_L, \perp_P, \perp\}$. We call such a sequence a valid out-of-sync sequence.

First let us consider what happens when event $E$ occurs. Notice that since the environment $\mathcal{I}$ provides for $\mathcal{A}$ is indistinguishable from that in $\mathcal{A}$'s real security game, if $\mathcal{A}$ submits a valid out-of-sync sequence of decryption queries, then $\mathcal{I}$ will have created a valid ciphertext forgery in its security game. The following therefore holds:

$$\Pr[\mathcal{A} \text{ wins} \wedge E] \leq \Pr[E] = \Pr[\mathcal{I} \text{ wins}] \leq \mathbf{Adv}^{\text{int-bsf-ctxt}}_{\text{SSH-CTR}[F]}.$$

Now let us consider what happens when $E$ does not occur. Notice that $\mathcal{B}$'s simulation of $\mathcal{A}$'s environment is correct so long as event $E$ does not occur. Thus, provided $E$ does not occur, if $\mathcal{A}$ is able to guess $b$ correctly then so does $\mathcal{B}$. The following therefore holds:

$$\Pr[\mathcal{A} \text{ wins} \wedge \neg E] \leq \Pr[\mathcal{B} \text{ wins}] \leq \frac{1}{2}\mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{SSH-CTR}[F]} + \frac{1}{2}.$$

**Adversary** $\mathcal{B}^{\mathcal{E}\text{-SSH-CTR}_{K_e,K_t}(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{L}(\cdot)}$
  $\texttt{cbuff} \leftarrow \varepsilon$; $st_d \leftarrow \varepsilon$
  Run $\mathcal{A}$
  **if** $\mathcal{A}$ makes an encryption query $m_0, m_1$ **then**
    Respond with $\mathcal{E}$-SSH-CTR$_{K_e,K_t}(\mathcal{LR}(m_0, m_1, b))$ to $\mathcal{A}$
  **end if**
  **if** $\mathcal{A}$ makes a decryption query $c$ **then**
    **if** $st_d = \perp$ **then**
      Respond with $\perp$ to $\mathcal{A}$
    **end if**
    $\texttt{cbuff} \leftarrow \texttt{cbuff} \| c$
    **if** $LF = \varepsilon$ **and** $|\texttt{cbuff}| \geq L$ **then**
      Parse cbuff as $\tilde{c} \| A$, where $|\tilde{c}| = L$
      $LF \leftarrow \mathcal{L}(\tilde{c})$
      **if** $LF = \perp_L$ **then**
        $st_d \leftarrow \perp$
        Respond with $\perp_L$ to $\mathcal{A}$
      **else**
        need $= 4 + LF +$ maclen
      **end if**
    **end if**
    **if** $|\texttt{cbuff}| \geq L$ **then**
      **if** $|\texttt{cbuff}| \geq$ need **then**
        **if** the query was in-sync **then**
          $LF \leftarrow \varepsilon$
          Parse cbuff as $A \| B$, where $|A| =$ need
          $\texttt{cbuff} \leftarrow B$
        **else**
          $st_d \leftarrow \perp$
          Respond with $\perp_A$ to $\mathcal{A}$
        **end if**
      **end if**
    **end if**
  **end if**
  Until $\mathcal{A}$ halts and returns a guess $b'$
  **return** $b'$

**Adversary** $\mathcal{I}^{\mathcal{E}\text{-SSH-CTR}_{K_e,K_t}(\cdot),\mathcal{D}\text{-SSH-CTR}_{K_e,K_t}(\cdot)}(k)$
  $b \xleftarrow{r} \{0,1\}$
  Run $\mathcal{A}$
  **if** $\mathcal{A}$ makes an encryption query $(m_0, m_1)$ **then**
    Respond with $\mathcal{E}$-SSH-CTR$_{K_e,K_t}(m_b)$
  **end if**
  **if** $\mathcal{A}$ makes an decryption query $c$ **then**
    Respond with $\mathcal{D}$-SSH-CTR$_{K_e,K_t}(c)$
  **end if**

**Fig. 7.** Adversaries for proof of Theorem 2

Now we complete the proof:

$$\begin{aligned}\mathbf{Adv}^{\text{lor-bsf-cca}}_{\text{SSH-CTR}[F]} &= 2\Pr[\mathcal{A} \text{ wins}] - 1 \\ &= 2\Pr[\mathcal{A} \text{ wins} \wedge E] + 2\Pr[\mathcal{A} \text{ wins} \wedge \neg E] - 1 \\ &\leq 2\mathbf{Adv}^{\text{int-bsf-ctxt}}_{\text{SSH-CTR}[F]} + \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{SSH-CTR}[F]}.\end{aligned}$$

□

**Proof of Lemma 1:** Consider an adversary $\mathcal{I}$ attacking SSH-CTR$[F]$ in the INT-BSF-CTXT sense. We will use this adversary to construct a new adversary $\mathcal{F}$ which will attack $\mathcal{MA}$ in the SUF-CMA sense. The construction of $\mathcal{F}$ is shown in Figure 8. We assume that $\mathcal{F}$'s algorithms enc and dec maintain sequence numbers $SN_e$, $SN_d$ for encryption and decryption.

It is evident that the simulation provided to $\mathcal{I}$ by $\mathcal{F}$ is correct. Suppose $\mathcal{I}$ wins the INT-BSF-CTXT game by making an out-of-sync sequence of decryption queries that does not result in an output in the set $\{\perp_L, \perp_A, \perp_P, \perp\}$. Then the corresponding ciphertext $\bar{c}[1 \ldots n] \| \tau$ processed by $\mathcal{F}$'s simulation of the decryption oracle was not output by $\mathcal{F}$'s simulation of the encryption oracle. Moreover, in this case, $\mathcal{F}$ makes a certain query $(m_t, \tau)$ to its verification oracle for which $v = 1$ during its simulation of decryption. We claim that this query ensures that $\mathcal{F}$ wins its forgery game.

**Adversary** $\mathcal{F}^{\mathcal{T}_{K_t}(\cdot),\mathcal{V}_{K_t}(\cdot)}$
  $K_e \xleftarrow{r} \mathcal{K}_e$; $st_e \leftarrow \varepsilon$; $st_d \leftarrow \varepsilon$; $m_e \leftarrow \varepsilon$
  Run $\mathcal{I}$
  **if** $\mathcal{I}$ makes an encryption query $m$ **then**
    **if** $st_e = \perp$ **then**
      Respond with $\perp$ to $\mathcal{I}$
    **end if**
    $(m_e, m_t) \leftarrow \mathsf{enc}(m)$
    **if** $m_e = \perp$ **then**
      $st_e \leftarrow \perp$
      Respond with $\perp$ to $\mathcal{I}$
    **end if**
    $\tau \leftarrow \mathcal{T}_{K_t}(m_t)$
    $c \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_e)$
    Respond with $c \| \tau$ to $\mathcal{I}$
  **end if**
  **if** $\mathcal{I}$ makes a decryption query $c$ **then**
    **if** $st_d = \perp$ **then**
      Respond with $\perp$ to $\mathcal{I}$
    **end if**
    $\mathtt{cbuff} \leftarrow \mathtt{cbuff} \| c$
    **if** $m_e = \varepsilon$ **and** $|\mathtt{cbuff}| \geq L$ **then**
      Parse $\mathtt{cbuff}$ as $\tilde{c} \| A$, where $|\tilde{c}| = L$
      $m_e[1] \leftarrow \mathcal{D}\text{-CTR}_{K_e}(\tilde{c})$
      $LF \leftarrow \mathsf{len}(m_e[1])$
      **if** $LF = \perp_L$ **then**
        $st_d \leftarrow \perp$
        Respond with $\perp_L$ to $\mathcal{I}$
      **end if**
      $\mathtt{need} = 4 + LF + \mathtt{maclen}$
    **end if**
    **if** $|\mathtt{cbuff}| \geq L$ **then**
      **if** $|\mathtt{cbuff}| \geq \mathtt{need}$ **then**
        Parse $\mathtt{cbuff}$ as $\bar{c}[1 \ldots n] \| \tau \| B$,
        where $|\bar{c}[1 \ldots n] \| \tau| = \mathtt{need}$ and $|\tau| = \mathtt{maclen}$
        $m_e[2 \ldots n] \leftarrow \mathcal{D}\text{-CTR}_{K_e}(\bar{c}[2 \ldots n])$
        $m_e \leftarrow m_e[1] \| m_e[2 \ldots n]$
        $m_t \leftarrow SN_d \| m_e$
        $v \leftarrow \mathcal{V}_{K_t}(m_t, \tau)$
        $m \leftarrow dec(m_e)$
        **if** ($v = 1$ **and** the sequence of queries in $\mathtt{cbuff}$ was in-sync) **then**
          $\mathtt{cbuff} \leftarrow B$
          $m_e \leftarrow \varepsilon$
        **else if** ($v = 1$ **and** the sequence of queries in $\mathtt{cbuff}$ was out-of-sync) **then**
          **if** $m = \perp_P$ **then**
            $st_d \leftarrow \perp$
          **end if**
          Respond with $m$ to $\mathcal{I}$
        **else**
          $st_d \leftarrow \perp$
          Respond with $\perp_A$ to $\mathcal{I}$
        **end if**
      **end if**
    **end if**
  **end if**

**Fig. 8.** Adversary for proof of Lemma 1

To justify this claim, we need to show that either this query is on a message that $\mathcal{F}$ did not query to its tagging oracle $\mathcal{T}_{K_t}(m_t)$ when preparing its responses to encryption queries, or that it is a new MAC tag on a message that $\mathcal{F}$ queried previously to $\mathcal{T}_{K_t}(m_t)$. Now, there are two cases, depending

**Experiment** $\mathbf{ExpH}_x$
$K_e \xleftarrow{r} \mathcal{K}_e; K_t \xleftarrow{r} \mathcal{K}_t; ctr \xleftarrow{r} \{0,1\}^l$
$st_0 \leftarrow \varepsilon; st_1 \leftarrow \varepsilon; st_2 \leftarrow \varepsilon$
$SN_0 \leftarrow 0; SN_1 \leftarrow 0; SN_1 \leftarrow 0$
Run $\mathcal{S}$
**if** $\mathcal{S}$ makes a length oracle query $c$ **then**
  Respond with $\mathcal{L}(c)$ to $\mathcal{S}$
**end if**
**if** $\mathcal{S}$ makes an $\mathcal{E}$-SSH-CTR query $(m_0, m_1)$ **then**
  $(m_{e,0}, m_{t,0}, st_0, SN_0) \leftarrow \mathsf{enc}^*(m_0, st_0, SN_0)$
  $(m_{e,1}, m_{t,1}, st_1, SN_1) \leftarrow \mathsf{enc}^*(m_1, st_1, SN_1)$
  $(m'_{e,1}, m'_{t,1}, st_2, SN_2) \leftarrow \mathsf{enc}^*(m_1, st_2, SN_2)$
  **switch** $(x)$:
    case $x = 0$: $\sigma \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_{e,1}); \tau \leftarrow \mathcal{T}_{K_t}(m_{t,1})$
    case $x = 1$: $\sigma \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_{e,1}); \tau \leftarrow \mathcal{T}_{K_t}(m'_{t,1})$
    case $x = 2$: $\sigma \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_{e,0}); \tau \leftarrow \mathcal{T}_{K_t}(m'_{t,1})$
    case $x = 3$: $\sigma \leftarrow \mathcal{E}\text{-CTR}_{K_e}(m_{e,0}); \tau \leftarrow \mathcal{T}_{K_t}(m_{t,0})$
  Respond with $\sigma \| \tau$ to $\mathcal{S}$
**end if**
Until $\mathcal{S}$ halts and returns a bit $b'$
**return** $b'$

**Fig. 9.** Experiment $\mathbf{ExpH}_x$ used in the proof of Lemma 2

on whether $\mathcal{I}$'s sequence of queries (as represented by the contents of `cbuff`) is current state or future state.

In the former case, since the query is out-of-sync, we know that $\bar{c}[1 \ldots n] \| \tau$ was not output by the encryption oracle for the same value of the sequence number, $SN_d$. It is then easy to see that either the plaintext $m_e$ differs from that encrypted, or the MAC tag $\tau$ differs from that output by the encryption oracle for the value $SN_d$. But that value $SN_d$ was only used in one query to the encryption oracle, and hence only used in making a single query to $\mathcal{T}_{K_t}(m_t)$ (here we use the fact that $q_e \leq 2^{32}$). Hence $\tau$ is either a valid MAC tag on a new message $m_t = SN_d \| m_e$, or a new MAC tag on the existing message $m_t$.

In the latter case, since the query is future state, we can deduce that the value $SN_d$ used in constructing the message $m_t = SN_d \| m_e$ has never been used in handling an encryption oracle query, and hence no message of the form $m_t = SN_d \| m_e$ has been queried by $\mathcal{F}$ to $\mathcal{T}_{K_t}(m_t)$ (here we use the fact that $q_d \leq 2^{32}$). Then $\tau$ is a valid MAC tag on a new message $m_t$.

Notice that in this analysis, $\mathcal{F}$ is still successful in winning its game even if $\mathcal{I}$'s out-of-sync decryption query results in an output $m = \perp_P$. So $\mathcal{F}$'s success probability is at least as great as that of $\mathcal{I}$. In the simulation, $\mathcal{F}$ makes at most $q_e$ queries to its tagging oracle and at most $q_d$ queries to its verification oracle. Also, a routine calculation shows that the total numbers of bits in $\mathcal{F}$'s queries are tightly related to those in $\mathcal{I}$'s, as in the statement of the lemma. □

**Proof of Lemma 2:** In the following proof, we let $\mathsf{enc}^*(\cdot, \cdot, \cdot)$ denote the encoding algorithm $\mathsf{enc}(\cdot)$ having some state $(st, SN)$ as part of its input and having the new state returned as output. We let $\mathcal{S}$ denote an LOR-LLSF-CPA adversary that has oracle access to the encryption oracle $\mathcal{E}\text{-SSH-CTR}_{K_e,K_t}(\mathcal{LR}(\cdot, \cdot, b))$, $b \in \{0,1\}$ and length oracle $\mathcal{L}(\cdot)$. In Figure 9 we define experiments $\mathbf{ExpH}_x$ for $x \in \{0,1,2,3\}$. Notice that in the experiments $(m_{e,1}, m_{t,1})$ and $(m'_{e,1}, m'_{t,1})$ will be identical except possibly in the random bytes used for padding.

Let $P_x = \Pr[\mathbf{ExpH}_x = 1]$, for $x \in \{0,1,2,3\}$. By the definition of $\mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{SSH-CTR}[F],\mathcal{S}}(k)$, we have

$$\mathbf{Adv}^{lor-llsf-cpa}_{\text{SSH-CTR}[F],\mathcal{S}}(k) = P_0 - P_3 = (P_0 - P_1) + (P_1 - P_2) + (P_2 - P_3).$$

We proceed by estimating the terms $P_i - P_{i+1}$, $i = 0, 1, 2$.

Given $\mathcal{S}$, we construct three new adversaries $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$, using their oracles to provide simulations of $\mathcal{S}$'s oracles, as shown in Figure 10. Both $\mathcal{A}_0$ and $\mathcal{A}_2$ will attack $\mathcal{MA}$ in the LOR-

**Adversary** $\mathcal{A}_0{}^{\mathcal{T}_K(\mathcal{LR}(\cdot,\cdot,b))}$
$K_e \xleftarrow{r} \mathcal{K}_e$; $st_1 \leftarrow \varepsilon$; $st_2 \leftarrow \varepsilon$; $st_l \leftarrow \varepsilon$
$SN_1 \leftarrow 0$; $SN_2 \leftarrow 0$; $SN_l \leftarrow 0$; $st_l \leftarrow \varepsilon$
$ctr_e \xleftarrow{r} \{0,1\}^l$; $ctr_l \leftarrow ctr_e$
Run $\mathcal{S}$
**if** $\mathcal{S}$ makes an encryption query $(m_0, m_1)$ **then**
  $(m_{e,1}, m_{t,1}, st_1, SN_1) \leftarrow \mathsf{enc}^*(m_1, st_1, SN_1)$
  $(m'_{e,1}, m'_{t,1}, st_2, SN_2) \leftarrow \mathsf{enc}^*(m_1, st_2, SN_2)$
  Parse $m_{e,1}$ as $m_{e,1}[1]\|m_{e,1}[2]\|\ldots\|m_{e,1}[n]$
  **for** $i = 1$ to $n$ **do**
    $\sigma[i] \leftarrow m_{e,1}[i] \oplus F_{K_e}(ctr_e + i)$
  **end for**
  $\tau \leftarrow \mathcal{T}_K(\mathcal{LR}(m'_{t,1}, m_{t,1}, b))$
  $ctr_e \leftarrow ctr_e + n$
  Respond with $\sigma[1]\|\sigma[2]\|\ldots\|\sigma[n]\|\tau$ to $\mathcal{S}$
**end if**
**if** $\mathcal{S}$ makes a length oracle query $c$ **then**
  **if** $st_l = \perp$ **then**
    Respond with $\perp$ to $\mathcal{S}$
  **end if**
  $m \leftarrow F_{K_e}(ctr_l) \oplus c$
  $LF \leftarrow \mathsf{len}(m)$
  **if** $LF \neq \perp_L$ **then**
    $ctr_l \leftarrow ctr_l + ((LF + 4)/L)$
    $SN_l \leftarrow SN_l + 1$
  **end if**
  **if** the query $c$ is out-of-sync **then**
    $st_l \leftarrow \perp$
  **end if**
  Respond with $LF$ to $\mathcal{S}$
**end if**
Until $\mathcal{S}$ halts and outputs $b'$
**return** $b'$

**Adversary** $\mathcal{A}_2{}^{\mathcal{T}_K(\mathcal{LR}(\cdot,\cdot,b))}$
$K_e \xleftarrow{r} \mathcal{K}_e$; $st_0 \leftarrow \varepsilon$; $st_2 \leftarrow \varepsilon$; $st_l \leftarrow \varepsilon$
$SN_0 \leftarrow 0$; $SN_2 \leftarrow 0$; $SN_l \leftarrow 0$; $st_l \leftarrow \varepsilon$
$ctr_e \xleftarrow{r} \{0,1\}^l$; $ctr_l \leftarrow ctr_e$
Run $\mathcal{S}$
**if** $\mathcal{S}$ makes an encryption query $(m_0, m_1)$ **then**
  $(m_{e,0}, m_{t,0}, st_0, SN_0) \leftarrow \mathsf{enc}^*(m_1, st_0, SN_0)$
  $(m'_{e,1}, m'_{t,1}, st_2, SN_2) \leftarrow \mathsf{enc}^*(m_1, st_2, SN_2)$
  Parse $m_{e,0}$ as $m_{e,0}[1]\|m_{e,0}[2]\|\ldots\|m_{e,0}[n]$
  **for** $i = 1$ to $n$ **do**
    $\sigma[i] \leftarrow m_{e,0}[i] \oplus F_{K_e}(ctr_e + i)$
  **end for**
  $\tau \leftarrow \mathcal{T}_K(\mathcal{LR}(m_{t,0}, m'_{t,1}, b))$
  $ctr_e \leftarrow ctr_e + n$
  Respond with $\sigma[1]\|\sigma[2]\|\ldots\|\sigma[n]\|\tau$ to $\mathcal{S}$
**end if**
**if** $\mathcal{S}$ makes a length oracle query $c$ **then**
  **if** $st_l = \perp$ **then**
    Respond with $\perp$ to $\mathcal{S}$
  **end if**
  $m \leftarrow F_{K_e}(ctr_l) \oplus c$
  $LF \leftarrow \mathsf{len}(m)$
  **if** $LF \neq \perp_L$ **then**
    $ctr_l \leftarrow ctr_l + ((LF + 4)/L)$
    $SN_l \leftarrow SN_l + 1$
  **end if**
  **if** the query $c$ is out-of-sync **then**
    $st_l \leftarrow \perp$
  **end if**
  Respond with $LF$ to $\mathcal{S}$
**end if**
Until $\mathcal{S}$ halts and outputs $b'$
**return** $b'$

**Adversary** $\mathcal{A}_1{}^{\mathcal{E}\text{-CTR}^{\mathcal{EC}}_{K_c}(\mathcal{LR}(\cdot,\cdot,b)),\mathcal{L}(\cdot)}$
$K_t \xleftarrow{r} \mathcal{K}_t$; $st_2 \leftarrow \varepsilon$; $SN_2 \leftarrow 0$
Run $\mathcal{S}$
**if** $\mathcal{S}$ makes a length oracle query $c$ **then**
  Respond with $\mathcal{L}(c)$ to $\mathcal{S}$
**end if**
**if** $\mathcal{S}$ makes an encryption query $(m_0, m_1)$ **then**
  $(m'_{e,1}, m'_{t,1}, st_2, SN_2) \leftarrow \mathsf{enc}^*(m_1, st_2, SN_2)$
  $\sigma \leftarrow \mathcal{E}\text{-CTR}^{\mathcal{EC}}_{K_e}(\mathcal{LR}(m_0, m_1, b))$; $\tau \leftarrow \mathcal{T}_{K_t}(m'_{t,1})$
  Respond with $\sigma\|\tau$ to $\mathcal{S}$
**end if**
Until $\mathcal{S}$ halts and outputs $b'$
**return** $b'$

**Fig. 10.** Adversaries used in the proof of Lemma 2

DCPA sense, while $\mathcal{A}_1$ will attack $\text{CTR}^{\mathcal{EC}}[F]$ in the LOR-LLSF-CPA sense. We will show that the following equalities hold:

$$P_0 - P_1 = \mathbf{Adv}^{\text{lor-dcpa}}_{\mathcal{MA},\mathcal{A}_0}(k),$$
$$P_1 - P_2 = \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}_1}(k),$$
$$P_2 - P_3 = \mathbf{Adv}^{\text{lor-dcpa}}_{\mathcal{MA},\mathcal{A}_2}(k).$$

Let us begin by proving the first equality. In the construction of $\mathcal{A}_0$, the adversary maintains counter $ctr_l$ and sequence number $SN_l$ for handling length oracle queries, and two sets of states

for handling encryption queries. In the construction, if the hidden bit $b$ equals 0 then $\mathcal{A}_0$ runs $\mathcal{S}$ in the environment provided by $\mathbf{ExpH}_1$, while if $b = 1$, then $\mathcal{A}_0$ runs $\mathcal{S}$ in the environment provided by $\mathbf{ExpH}_0$. On the other hand, it is clear that, because of the use of distinct sequence numbers in constructing the messages $m_{t,1}, m'_{t,1}$ when handling encryption queries, and the limitation that $q_e \leq 2^{32}$, $\mathcal{A}_0$ is a valid LOR-DCPA adversary, outputting 1 whenever $\mathcal{S}$ does. Putting all of this together, we see that

$$\begin{aligned}\mathbf{Adv}^{\text{lor-dcpa}}_{\mathcal{MA},\mathcal{A}_0}(k) &= \Pr[\mathbf{Exp}^{\text{lor-dcpa-1}}_{\mathcal{MA},\mathcal{A}_0}(k) = 1] - \Pr[\mathbf{Exp}^{\text{lor-dcpa-0}}_{\mathcal{MA},\mathcal{A}_0}(k) = 1] \\ &= \Pr[\mathbf{ExpH}_0 = 1] - \Pr[\mathbf{ExpH}_1 = 1] \\ &= P_0 - P_1.\end{aligned}$$

Next, we prove the second equality. In $\mathcal{A}_1$'s construction, if the hidden bit $b$ equals 0 then $\mathcal{A}_1$ runs $\mathcal{S}$ in the environment provided by $\mathbf{ExpH}_2$, while if $b = 1$, then $\mathcal{A}_1$ runs $\mathcal{S}$ in the environment provided by $\mathbf{ExpH}_1$. On the other hand, it is clear that $\mathcal{A}_1$ is a valid LOR-LLSF-CPA adversary against the combined scheme $\text{CTR}^{\mathcal{EC}}[F]$, outputting 1 whenever $\mathcal{S}$ does. Combining these facts, we have:

$$\begin{aligned}\mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}_1}(k) &= \Pr[\mathbf{Exp}^{\text{lor-llsf-cpa-1}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}_1}(k) = 1] - \Pr[\mathbf{Exp}^{\text{lor-llsf-cpa-0}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}_1}(k) = 1] \\ &= \Pr[\mathbf{ExpH}_1 = 1] - \Pr[\mathbf{ExpH}_2 = 1] \\ &= P_1 - P_2.\end{aligned}$$

The third and final equality follows using a simulation similar to the first. For completeness, we give full details. In the construction of $\mathcal{A}_2$ counter $ctr_l$ and sequence number $SN_l$ are maintained for handling length oracle queries, and two sets of state for handling encryption queries. In the construction, if the hidden bit $b$ equals 0 then $\mathcal{A}_2$ runs $\mathcal{S}$ in the environment provided by $\mathbf{ExpH}_3$, while if $b = 1$, then $\mathcal{A}_2$ runs $\mathcal{S}$ in the environment provided by $\mathbf{ExpH}_2$. On the other hand, it is clear that, because of the use of distinct sequence numbers in constructing the messages $m_{t,0}, m'_{t,1}$ when handling encryption queries, and the limitation that $q_e \leq 2^{32}$, $\mathcal{A}_2$ is a valid LOR-DCPA adversary, outputting 1 whenever $\mathcal{S}$ does. Putting all of this together, we see that

$$\begin{aligned}\mathbf{Adv}^{\text{lor-dcpa}}_{\mathcal{MA},\mathcal{A}_2}(k) &= \Pr[\mathbf{Exp}^{\text{lor-dcpa-1}}_{\mathcal{MA},\mathcal{A}_2}(k) = 1] - \Pr[\mathbf{Exp}^{\text{lor-dcpa-0}}_{\mathcal{MA},\mathcal{A}_2}(k) = 1] \\ &= \Pr[\mathbf{ExpH}_2 = 1] - \Pr[\mathbf{ExpH}_3 = 1] \\ &= P_2 - P_3.\end{aligned}$$

The lemma follows by accounting for the resources used by adversaries $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$ in terms of those used by $\mathcal{S}$.

□

**Proof of Lemma 3:** Assume that $\mathcal{A}$ attacks the scheme $\text{CTR}^{\mathcal{EC}}[F]$ in the LOR-LLSF-CPA sense. Let $\text{CTR}^{\mathcal{EC}}[R]$ denote the same scheme but with $F$, the prf family, replaced by $R$, the set of random functions mapping $l$-bit strings to $L$-byte strings. We build a distinguisher $\mathcal{D}$ against the prf family as follows: $\mathcal{D}$, with oracle access to a function $f$ drawn from either $F$ or $R$, runs $\mathcal{A}$ and sees whether $\mathcal{A}$ breaks the scheme. If so it guesses that $f$ was drawn from the prf family $F$, otherwise it guesses that $f$ was drawn from the set of all functions $R = Rand^{l\to L}$. $\mathcal{D}$ simulates $\mathcal{A}$'s oracles $\mathcal{E}$-$\text{CTR}^{\mathcal{EC}}[F]$ and $\mathcal{L}$ by making queries to $f$. The construction is shown in Figure 11.

We now examine the advantage of $\mathcal{D}$. When $f$ is drawn from $F$, then $\mathcal{D}$ runs $\mathcal{A}$ in an environment where $\mathcal{A}$ plays against the scheme $\text{CTR}^{\mathcal{EC}}[F]$, while when $f$ is drawn from $R$, then $\mathcal{D}$ runs $\mathcal{A}$ in an environment where $\mathcal{A}$ plays against the scheme $\text{CTR}^{\mathcal{EC}}[R]$. Moreover, by a standard argument $\mathcal{A}$ correctly guesses $b' = b$ with probability either $\frac{1}{2}(1 + \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}}(k))$ or

**Algorithm** $\mathcal{D}^f$
$b \xleftarrow{r} \{0,1\}$
$st_e \leftarrow \varepsilon$; $ctr_e \xleftarrow{r} \{0,1\}^l$; $ctr_l \leftarrow ctr_e$; $SN_l \leftarrow 0$;
$st_l \leftarrow \varepsilon$
Run $\mathcal{A}$
**if** $\mathcal{A}$ makes an encryption query $(m_0, m_1)$ **then**
  **if** $st_e = \perp$ **then**
    Respond with $\perp$ to $\mathcal{A}$
  **end if**
  $(m_{e,b}, m_{t,b}) \leftarrow \mathsf{enc}(m_b)$
  **if** $m_{e,b} = \perp$ **then**
    $st_e \rightarrow \perp$
    Respond with $\perp$ to $\mathcal{A}$
  **end if**
  Parse $m_{e,b}$ as $m_{e,b}[1]m_{e,b}[2]\ldots m_{e,b}[n]$
  **for** $i = 1$ to $n$ **do**
    $c[i] \leftarrow m_{e,b}[i] \oplus f(ctr_e + i)$
  **end for**
  $ctr_e \leftarrow ctr_e + n$
  Respond with $c[1]c[2]\ldots c[n]$ to $\mathcal{A}$
**end if**
**if** $\mathcal{A}$ makes a length oracle query $c$ **then**
  **if** $st_l = \perp$ **then**
    Respond with $\perp$ to $\mathcal{A}$
  **end if**
  $m \leftarrow f(ctr_l) \oplus c$
  $LF \leftarrow \mathsf{len}(m)$
  **if** $LF \neq \perp_L$ **then**
    $ctr_l \leftarrow ctr_l + ((LF + 4)/L)$
    $SN_l \leftarrow SN_l + 1$
  **end if**
  **if** query $c$ is out-of-sync **then**
    $st_l \leftarrow \perp$
  **end if**
  Respond with $LF$ to $\mathcal{S}$
**end if**
Until $\mathcal{A}$ halts and outputs $b'$
**if** $b' = b$ **then**
  **return** 1
**else**
  **return** 0
**end if**

**Fig. 11.** Distinguisher used in the proof of Lemma 3

$\frac{1}{2}(1 + \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[R],\mathcal{A}}(k))$, depending on the environment supplied by $\mathcal{D}$. It then follows that:

$$\begin{aligned}
\mathbf{Adv}^{\text{prf}}_{F,\mathcal{D}}(k) &= \Pr[\text{guess } f \leftarrow F | f \leftarrow F] - \Pr[\text{guess } f \leftarrow F | f \leftarrow R] \\
&= \Pr[b' = b | f \leftarrow F] - \Pr[b' = b | f \leftarrow R] \\
&= \tfrac{1}{2}(1 + \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}}(k)) - \tfrac{1}{2}(1 + \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[R],\mathcal{A}}(k)) \\
&= \tfrac{1}{2}(\mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[F],\mathcal{A}}(k) - \mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[R],\mathcal{A}}(k)).
\end{aligned}$$

The result then follows after rearranging the above expression and accounting for the number of queries made to $f$ in $\mathcal{D}$'s simulation. □

**Proof of Lemma 4:** First we argue that the $\mathcal{L}$ oracle provides the adversary with no new information. We remove the abstraction of the length revealing oracle, and consider what operation the oracle is performing. The length oracle acts as a partial decryption oracle for the first block.

The function $f$ is queried with the current counter value $ctr_l$. The output of $f(ctr_l)$ is then xored with a ciphertext block to obtain the plaintext block from which the length field can be read.

We must consider three different situations: when an adversary makes an in-sync query, a current state out-of-sync query or a future state out-of-sync query. Recall that we only allow an adversary to make one out-of-sync query.

First consider the situation that an adversary $\mathcal{A}$ queries $\mathcal{L}$ with an in-sync query, i.e. the first block output by the encryption oracle for the same sequence number. The adversary already knows the output since it knows the length of the encoded messages sent to the encryption oracle. The adversary therefore gains no new information from this type of query.

Next consider the situation that an adversary $\mathcal{A}$ queries $\mathcal{L}$ with a current state out-of-sync query, i.e. before being queried, $\mathcal{L}$ and the encryption oracle have the same counter value $ctr_l = ctr_e$ but the input to $\mathcal{L}$ is different to the first block of output from the encryption oracle. Let $(m_0, m_1)$ be $\mathcal{A}$'s encryption oracle query with corresponding ciphertext output $c_b = \mathcal{E}\text{-CTR}_K^{\mathcal{EC}}(m_b)$. Let $c$ be $\mathcal{A}$'s length oracle query, such that $c \neq c_b[1]$. The length oracle must first decrypt $c$ to obtain the plaintext block $m = c \oplus f(ctr_l)$ from which the length field can be obtained and checked. Based on the response from the length oracle the adversary can determine partial knowledge of $m$. If the length field $LF$ is returned the adversary then knows the first 4 bytes of $m$. Otherwise if $\perp_L$ was returned the adversary may still be able to determine some information about the length field based on the specific length checks performed. We know that $c = m \oplus f(ctr_l)$, and this allows the adversary to deduce information about the first 4 bytes of $f(ctr_l)$. Due to $f$ being a truly random function this partial knowledge of $f(ctr_l)$ does not allow the adversary to determine any further bits of $f(ctr_l)$, nor any information about $f(ctr')$ for any value $ctr' \neq ctr_l$. Thus the adversary gains no information about the encrypted message other than what can be deduced from its partial knowledge of $f(ctr_l)$. But it is clear from the decryption equation $\mathsf{enc}(m_b)[1] = c_b[1] \oplus f(ctr_l)$ that the information that the adversary obtains concerning the encrypted message will be confined to the first four bytes of $\mathsf{enc}(m_b)$. Since the first four bytes of $\mathsf{enc}(m_b)$ contains the packet length field and this is identical for both $m_0$ and $m_1$ and is already known to the adversary (since it can be determined based on the lengths of the query $(m_0, m_1)$), the adversary gains no new information from this type of query.

Finally consider the situation that an adversary $\mathcal{A}$ queries $\mathcal{L}$ with a future state out-of-sync query, i.e. the $\mathcal{L}$ oracle has a more advanced counter value than the encryption oracle. Let the value of the counter held at the $\mathcal{L}$ oracle at the start of the $\mathcal{L}$ query be $ctr_l$. The adversary makes a query $c$ to $\mathcal{L}$. Since an adversary is only allowed one out-of-sync query this implies that the next encryption oracle query must start with the same counter value $ctr_l = ctr_e$. Again, the adversary gains partial information about $m = c \oplus f(ctr_l)$ (information about the length field, i.e. the first four bytes of $m$), he can again use this to determine partial information about $f(ctr_l)$. Due to $f$ being a truly random function this partial knowledge of $f(ctr_l)$ does not allow the adversary to determine any further bits of $f(ctr_l)$, nor any information about $f(ctr')$ for any value $ctr' \neq ctr_l$. Next the adversary makes the encryption query $(m_0, m_1)$ with corresponding ciphertext output $c_b$. Since this encryption query will commence with the same counter value as in the length oracle query $c$, the adversary can again use his partial knowledge of $f(ctr_l)$ and knowledge of $c_b[1]$, to determine partial information about $\mathsf{enc}(m_b)[1] = c_b[1] \oplus f(ctr_l)$. The partial information that the adversary obtains is again confined to the first four bytes of $\mathsf{enc}(m_b)$. Since the first four bytes of $\mathsf{enc}(m_b)$ contains the packet length field and this is identical for both $m_0$ and $m_1$ and is already known to the adversary (since it can be determined based on the lengths of the query $(m_0, m_1)$), the adversary gains no new information from this type of query.

We must also consider the extreme case where the maximum number of bits have been queried to the encryption oracle. We restrict $\mu_e \leq 8L2^l - 8q_e(8 + L)$. This ensures that the counter $ctr_e$ cannot wrap around, taking into account the possibility that each query may be expanded by up to $8 + L$ bytes due to encoding. Assume that the maximum number of bits have been queried to

the encryption oracle and that all corresponding in-sync length oracle queries have been made. This implies that the next length oracle query will be a future state out-of-sync query. We now have two scenarios which we must consider: firstly that all possible counters have been queried and secondly that not all possible counters have been queried due to smaller expansion from encoding. In the first case the next length oracle query will be handled using a value $ctr_l$ that has wrapped around and is therefore equal to the counter value that was used at the start of the first encryption oracle query. By a similar argument to that above, the adversary will gain only part of $f(ctr_l)$ and has prior knowledge of any other data obtained due to observing the length of the corresponding encrypted messages. In the second case the next length oracle query will operate using a counter value $ctr_l$ that was never and will never be queried to the encryption oracle. Again the adversary will only be able to deduce 4 bytes of $f(ctr_l)$ and since there are no related ciphertexts there is no further plaintext that can be deduced. The adversary therefore gains no new information in either scenario.

By the argument above we know that adversary $\mathcal{A}$ gains no information from his access to the $\mathcal{L}$ oracle. Therefore the probability that an adversary wins the LOR-LLSF-CPA game is equal to the probability that an adversary wins the LOR-CPA game. It is then easy to see that the following equation holds:

$$\mathbf{Adv}^{\text{lor-llsf-cpa}}_{\text{CTR}^{\mathcal{EC}}[R]} = \mathbf{Adv}^{\text{lor-cpa}}_{\text{CTR}[R]}.$$

It therefore remains to prove that:

$$\mathbf{Adv}^{\text{lor-cpa}}_{\text{CTR}[R]} = 0.$$

The rest of the proof then proceeds as in the proof of Lemma 12 in [4], noting that our notation is slightly different from that in [4] because our functions are assumed to have $L$-byte outputs. □

**Proof of Lemma 5:** We wish to show that if $\mathcal{MA}$ is not secure, then $\mathcal{T}$, the tagging algorithm from $\mathcal{MA}$, is not a good prf family. Assume that $\mathcal{A}$ attacks $\mathcal{MA}$ in the LOR-DCPA sense with advantage greater than $\mathbf{Adv}^{\text{lor-dcpa}}_{\mathcal{MA}}$. We build a distinguisher $D$ using $\mathcal{A}$ with advantage better than $\mathbf{Adv}^{prf}_{\mathcal{T}}$, contradicting the assumed security of $\mathcal{T}$ as a prf family.

$D$ runs $\mathcal{A}$ and sees whether $\mathcal{A}$ breaks the scheme. If so it guesses that the tagging function was drawn from the prf family $\mathcal{T}$, otherwise it guesses that it was drawn from the set of all random functions with the same input and output size as $\mathcal{T}$. $D$ simulates $\mathcal{A}$'s tagging oracle queries by making queries to its oracle.

The rest of this proof is easy to construct and we omit the details. □