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,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 [4,11,12,13,14].

The SSH Binary Packet Protocol (BPP), as specified in [13], 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.* [3]. Bellare *et al.* introduced a stateful security model and notion for SSH-style protocols. They also proved that several minor variants of the SSH

* This research was supported in part by the European Commission under contract ICT-2007-216676 (ECRYPT-II).

** 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.

H. Gilbert (Ed.): EUROCRYPT 2010, LNCS 6110, pp. 345–361, 2010.

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 [3]. 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 [3]: 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 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 [7]. 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 [3] 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 [3] 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 [3], while it is assumed in [3] that ciphertexts are processed in an atomic fashion. Moreover, while the model of [3] 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 [3], 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 [3] 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 [3] 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 [10]. 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 [3] 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.

1.2 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 [13]. 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 [4]. 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.

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 [3], 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. At this point, our model begins to significantly diverge from the model of [3].

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 [13]:

> *"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 [13] 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 ε 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 enc and a decoding algorithm dec. The encoding algorithm enc is stateful and randomised, takes as input a message m and outputs two messages (m_e, m_t). Here as in [3], 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 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 dec is slightly different to that in [3] which had two outputs m and m_t. Note also that dec will 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 $\mathsf{dec}(m_e) \neq \perp_P$.

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. 1. Encoding Scheme for SSH

The specific encoding scheme used by the SSH BPP specification is shown in Figure 1. 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 enc, 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 [4] 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 [3], the equivalent state consists of a single value which is "over-loaded" to carry both the algorithm status and sequence number. For concreteness, Figure 1 shows the specific parsing steps carried out by OpenSSH during decoding. 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 [3]. When we come to formally analyze the security of SSH-CTR, we will regard the block cipher as being a pseudorandom function (prf) family rather than as a pseudorandom permutation family. This allows us to directly use some of the results from [2]. Our definition for a prf family can be found in the full version of this paper [9].

We give a formal definition for counter mode encryption based on a prf family F, $\mathrm{CTR}[F] = (\mathcal{K}\text{-CTR}, \mathcal{E}\text{-CTR}, \mathcal{D}\text{-CTR})$ in [9]. The key generation algorithm $\mathcal{K}$-CTR outputs a random k-bit key K_e for the underlying prf 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 the encryption and decryption algorithms $\mathcal{E}$-CTR, $\mathcal{D}$-CTR.

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 1. Full details of this scheme appear in [9]. This construction is not used in SSH, but is needed as a step in our security analysis in Section 5.

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 τ. The verification algorithm $\mathcal{V}$, which is deterministic and stateless, takes as finput the key K_t and an encoded message m_t and a candidate tag τ' and outputs a bit. For any key K_t, message m_t and internal state of $\mathcal{T}_{K_t}$, we require that $\mathcal{V}_{K_t}(m_t, \mathcal{T}_{K_t}(m_t)) = 1$.

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 [3] 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 2. 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 keyspaces. 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 to the function len. This accords with our discussion in Section 2.1. The function len is shown as part of Figure 2. 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 2 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,

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
    $c \leftarrow \mathcal{E}$-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 $|\tilde{c}| = L$)
    $m_e[1] \leftarrow \mathcal{D}$-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 $|\tau| = \mathtt{maclen}$
      $m_e[2 \ldots n] \leftarrow \mathcal{D}$-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. 2. SSH-CTR, SSH using counter mode encryption

and fails if this is not the case. For further discussion, see the full version [9]. 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 and 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 1). 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 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 [3].

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 [2] 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 the full version [9]. We name our new model LOR-LLSF-CPA, where "LLSF" stands for "length leaking stateful".

In [3], 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:

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

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, μ_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* μ_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. 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 [3], the adversary is given 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:

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

The advantage function of the scheme is defined to be

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

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 μ_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 μ_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 [3]: 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 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 [3] 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.

In this INT-BSF-CTXT model, the adversary has access to encryption and decryption oracles, and is considered successful if it is able to make an out-of-sync sequence of decryption queries that results in an output from the decryption oracle that is not a member of the set $\{\perp_L, \perp_A, \perp_P, \perp\}$. 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. The formal definition of the INT-BSF-CTXT model can be found in the full version of this paper [9].

4.4 Security of Message Authentication Schemes

Finally, we define two security notions for MACs. We will use the LOR-DCPA notion from [3], 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 also be found in the full version [9].

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 prf family F and MAC scheme $\mathcal{MA}$ used in its construction. The structure of our proof follows that in [3], 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 are given in the full version of this paper [9].

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 for $q_e, q_d \leq 2^{32}$, $\mu_e \leq 8L2^l - 8q_e(8+L)$ and any t, k, μ_d, we have:*

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

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 [5]; 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 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 for $q_e, q_d \leq 2^{32}$ and any k, t, μ_e, μ_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 $\mu_v \leq \mu_d + 32q_d$.

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 for $q_e, q_l \leq 2^{32}$ and any k, t, μ_e, we have:*

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

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 prf 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, μ_e, q_l, we have:*

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

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:*

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

Lemma 5. *Let $\mathcal{MA}$ be a message authentication scheme. Then for any k, t and q_t, we have:*

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

where $t' = O(t)$.

6 Conclusion

We have extended the security model of Bellare *et al.* [3] 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 [3]. 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.

References

1. Albrecht, M.R., Paterson, K.G., Watson, G.J.: Plaintext recovery attacks against SSH. In: IEEE Symposium on Security and Privacy, pp. 16–26. IEEE Computer Society, Los Alamitos (2009)
2. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS 1997), pp. 394–403. IEEE, Los Alamitos (1997)

3. Bellare, M., Kohno, T., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. ACM Transactions on Information and Systems Security 7(2), 206–241 (2004)
4. Bellare, M., Kohno, T., Namprempre, C.: The Secure Shell (SSH) Transport Layer Encryption Modes. RFC 4344 (January 2006), http://www.ietf.org/rfc/rfc4344.txt
5. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
6. Canvel, B., Hiltgen, A.P., Vaudenay, S., Vuagnoux, M.: Password interception in a SSL/TLS channel. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 583–599. Springer, Heidelberg (2003)
7. CPNI Vulnerability Advisory. Plaintext recovery attack against SSH (November 14, 2008), http://www.cpni.gov.uk/Docs/Vulnerability_Advisory_SSH.txt (revised November 17, 2008)
8. Krawczyk, H.: The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer, Heidelberg (2001)
9. Paterson, K.G., Watson, G.J.: Plaintext-Dependent Decryption: A Formal Security Treatment of SSH-CTR. Cryptology ePrint Archive, Report 2010/095 (2010), http://eprint.iacr.org/2010/095
10. SSH usage profiling, http://www.openssh.org/usage/index.html
11. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Protocol Architecture. RFC 4251 (January 2006), http://www.ietf.org/rfc/rfc4251.txt
12. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Authentication Protocol. RFC 4252 (January 2006), http://www.ietf.org/rfc/rfc4252.txt
13. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (January 2006), http://www.ietf.org/rfc/rfc4253.txt
14. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Connection Protocol. RFC 4254 (January 2006), http://www.ietf.org/rfc/rfc4254.txt