Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol

Mihir Bellare
UC San Diego
mihir@cs.ucsd.edu

Tadayoshi Kohno
UC San Diego
tkohno@cs.ucsd.edu

Chanathip Namprempre*
Thammasat Univ, Thailand
meaw@alum.mit.edu

ABSTRACT

The Secure Shell (SSH) protocol is one of the most popular cryptographic protocols on the Internet. Unfortunately, the current SSH authenticated encryption mechanism is insecure. In this paper we propose several fixes to the SSH protocol and, using techniques from modern cryptography, we prove that our modified versions of SSH meet strong new chosen-ciphertext privacy and integrity requirements. Furthermore, our proposed fixes will require relatively little modification to the SSH protocol or to SSH implementations. We believe that our new notions of privacy and integrity for encryption schemes with stateful decryption algorithms will be of independent interest.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; E.3 [**Data Encryption**]: Standards; F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Security, Theory.

Keywords

Authenticated Encryption, Secure Shell, SSH, Stateful Decryption, Security Proofs.

1. Introduction

Conceived as a secure alternative to traditional Unix tools like `rsh` and `rcp` [27], the IETF standardization body's *Secure Shell* (*SSH*) protocol [17][1] has become one of the most popular and widely used cryptographic protocols on the Internet. Because of its popularity and because of the insecurity of programs like `rsh` and `telnet`, a number of institutions only allow users to remotely access their facilities using SSH. The cryptographic heart of the SSH protocol is its *Binary Packet Protocol* (*BPP*) [28]; the BPP is responsible for the underlying symmetric encryption and authentication (or the *authenticated encryption*) of all messages sent between two parties involved in an SSH connection.

Although others have discussed specific properties of the SSH BPP (e.g., problems with not using a MAC [26] or problems with SSH's use of CBC mode [10]), to the best of our knowledge no one has performed a rigorous, provable security-based analysis of the entire SSH BPP authenticated encryption mechanism. Our goal was thus to thoroughly analyze the SSH BPP authenticated encryption scheme and, in the event that we found any problems, to present provably-secure fixes to the protocol.

In order for our fixes to be as useful as possible to the Internet community, when developing our fixes we considered both (1) provable security and (2) efficiency. Additionally, since retroactively modifying existing implementations is often very expensive, we required that our suggested modifications (3) not significantly alter the current SSH specification. For the last point, we note that the creators of SSH had the foresight to design the SSH BPP in a modular way: in particular, it is relatively "easy" to change the SSH BPP's underlying encryption and message authentication modules.

ANALYSIS AND PROVABLY SECURE RECOMMENDATIONS. The SSH BPP specification states that SSH implementations should use CBC mode encryption [11] with chained initialization vectors (IVs); i.e., the IV used when encrypting a message should be the last block of the previous ciphertext. Unfortunately, CBC mode encryption with chained IVs is insecure [23], and this insecurity extends to SSH (this extension was also reported by Dai [10]).

Since CBC mode encryption with chained IVs is insecure, but CBC mode with random IVs is provably secure against chosen-plaintext attacks [2], a natural fix to the SSH protocol might be to replace the use of chained-IV CBC mode with randomized CBC mode. Unfortunately, we show that doing so is not sufficient. In particular, since the SSH specification does not require the padding to be random, the resulting SSH implementation may be vulnerable to a rather serious reaction-attack (a privacy attack that works by modifying a sender's ciphertexts and observing the receiver's response).

*Work done while at University of California, San Diego.

[1] SSH version 1.5 [27] and SSH version 2.0 [17] are very different (e.g., version 1.5 uses 32-bit CRCs for authenticity). We focus our analysis on SSH version 2.0.

CCS'02, November 18–22, 2002, Washington, DC, USA.

We next present several secure fixes to the SSH authenticated encryption mechanism. For example, we suggest using randomized CBC mode encryption; the difference between this suggestion and the suggestion in the above paragraph is that we require at least one full block of random padding (this could, however, result in having to encipher more blocks than the previous SSH alternative). We also suggest another CBC variant that does not require additional random padding: CBC mode where the IV is generated by encrypting a counter with a different key. As an additional alternative, we suggest replacing the underlying encryption scheme with a variant of counter (CTR) mode [12, 22] in which both the sender and receiver maintain a copy of the counter. We also present a framework within which to analyze other possible replacements.

One important advantage of these fixes over the current SSH specification is provable security. Making reasonable assumptions (e.g., that SSH's underlying block cipher is secure), we are able to show that our alternatives will preserve privacy against adaptive chosen-plaintext and adaptive chosen-ciphertext attacks. We also show that our alternatives will resist forgery, replay, and out-of-order delivery attacks. Finally, we argue that our alternatives, and especially the latter two, also satisfy the other two requirements listed above (efficiency and ease of modification). (We note that our CTR mode construction addresses the concerns with CTR mode raised in [8].)

THEORETICAL CONTRIBUTIONS. The previous notions of privacy [2] and integrity [19, 5] for authenticated encryption schemes only address encryption schemes with stateless decryption algorithms. The SSH BPP decryption algorithm is, however, stateful. Motivated by a desire to analyze the SSH BPP authenticated encryption scheme, and by the desire to capture the potential "power" of stateful decryption algorithms, we extend the previous notions of privacy and integrity to encryption schemes with stateful decryption algorithms. The aforementioned "power" refers to the fact that if a scheme meets our new notions of security, then, in addition to satisfying the existing notions of privacy [2] and integrity [19, 5], the scheme will be secure against replay attacks and out-of-order delivery attacks — attacks not captured under the previous models.

One alternative approach to our analysis would have been to model the SSH BPP as a "secure channel" [9] since the notion of secure channels can be applied to encryption schemes with stateful decryption algorithms. We point out that the combination of our notions is stronger than the notion of secure channels: combining a secure key agreement protocol with an authenticated encryption scheme that meets both of our notions will yield a secure channel. Consequently, since our fixes to the SSH BPP provably meet our strong notions, the resulting SSH BPP is also a secure channel.

We acknowledge that one potential disadvantage of our new notions of security is that they may be "too strong" for some applications: some applications may not require the strength associated with our notions (see [9, 20] for reasons). For those applications, the notion of a secure channel might be more appropriate. Our notions are, however, more appropriate for applications (like SSH) that do require a higher level of protection such as protection against out-of-order delivery attacks. Finally, we note that side-channel attacks (such as those exploiting information leaked through the length of packets or the interval of time between packets [25]) are not captured by our security models nor any other provable security models that we are aware of.

OVERVIEW. After describing the SSH Binary Packet Protocol in Section 2, we present a simple attack against the current SSH specification (Section 3). In Section 4 we show that "fixing" the SSH BPP in the natural way may result in an insecure protocol. Motivated by the lessons we learned from Sections 3 and 4, we then present provably-secure fixes to the SSH Binary Packet Protocol (Section 5). In Section 6 we present our provable security results in more detail (the proofs are in the full version of this paper [4]). Finally, in Section 7 we discuss our results and make recommendations to the SSH and applied cryptographic communities. We discuss the significance of our earlier attacks and the advantages and disadvantages of switching to our proposed modifications. We also discuss the possibility of changing the SSH BPP from an "Encrypt-*and*-MAC-based" construction to an "Encrypt-*then*-MAC-based" construction and the possibility of modifying SSH to use a dedicated authenticated encryption scheme such as XCBC [13] or OCB [24].

BACKGROUND AND RELATED WORK. An *authenticated encryption scheme* is a scheme designed to provide both privacy and integrity. From an API perspective, a symmetric authenticated encryption scheme is equivalent to an encryption scheme except that the decryption algorithm can return a special error code. There are two types of authenticated encryption schemes: dedicated constructions (e.g., RPC [19], XCBC [13], IACBC [18], and OCB [24]) and *generic composition* constructions, so named because they use standard encryption and message authentication schemes as "black boxes." Analysis of the latter class was initiated in [5, 20]. The schemes of SSH, SSL and IPSEC fall in this class. The idea of modeling data formats via encoding schemes that we use here was introduced in [6]. An et. al. [1] consider generic composition in the asymmetric setting and in particular obtain results about the security of the transform which splits a message into two sub-messages via a commitment scheme, signs one of the sub-messages and encrypts the other.

2. The SSH Binary Packet Protocol

The SSH Binary Packet Protocol [28] is responsible for encrypting and authenticating all messages between two parties involved in an SSH session. Before beginning the authenticated encryption portion of an SSH session, a client and a server first agree upon a set of shared symmetric keys (a different set for each direction of a connection). The client and the server also agree upon which encryption and message authentication schemes they wish to use. All of the encryption schemes recommended by [28] are based on CBC mode encryption [11], and all of the recommended message authentication schemes are based on HMAC [21].

The SSH authenticated encryption scheme works as shown in Figure 1. Given a *payload* message (in bytes), the SSH BPP encodes that message into an encoded packet consisting of the following fields: a four-byte packet length field containing the length of the remaining encoded packet (in bytes), a one-byte padding length field, the payload message, and (possibly random) padding. The length of the total packet must be a multiple of the underlying block cipher's block length, and the padding must be at least four bytes long. Although the SSH specification allows up to 255

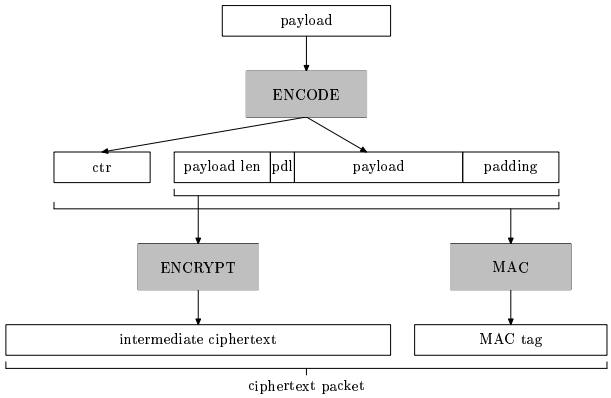

Figure 1: The SSH authenticated encryption scheme. See Section 2 for details.

bytes of padding per encoded packet, both implementations that we evaluated (`openssh-2.9p2` and SSH Communications' `ssh-3.0.1`) use the minimum padding necessary. The resulting ciphertext is the concatenation of the encryption of the above encoded packet and the MAC of the above encoded packet prepended with a 32-bit counter. In the following discussions we try to make clear whether we are referring to the *intermediate ciphertext* output by the underlying encryption scheme or the *ciphertext packet* (the concatenation of the intermediate ciphertext and the MAC tag) output by the SSH BPP.

Decryption is defined in a natural way: the receiver first decrypts the intermediate ciphertext portion of a ciphertext to get an encoded packet. The receiver then prepends a 32-bit counter (which it also maintains) to the encoded packet and determines whether the received MAC tag is valid. If so, the decryptor removes the payload from the encoded packet and delivers the payload to the user (or a higher-level protocol). If the MAC verification fails, the connection is terminated.

The SSH specification recommends the use of CBC mode with inter-packet chaining. This means that, when encrypting an encoded payload, the sender uses as the initialization vector (IV) either the last block of the immediately preceding ciphertext or, when encrypting the first message, an IV computed during the SSH key agreement protocol. We refer to the current instantiation of the SSH protocol as SSH-IPC, or SSH with inter-packet chaining.

3. Attack Against the Standard Implementation of SSH

There is a simple attack against SSH-IPC; this attack was also recently reported by Dai [10]. The problem with SSH-IPC is that an attacker will know the IV for the next message to be encrypted before the next message is actually encrypted. This means that if an attacker can control the entire first block of the input into SSH-IPC's underlying CBC encryption scheme, it will be able to control the corresponding input to the underlying block cipher. Since a block cipher is deterministic, an attacker could use this to glean information about a previously encrypted message.

We describe the attack in slightly more detail. We assume for now that an adversary can control the entire first block of an encoded packet. Suppose that an adversary has a guess G of the first encoded block of the ith packet, and let C_1 be the last CBC block of the $i-1$st intermediate ciphertext. Since we are considering SSH-IPC, the block C_1 was used as the IV when encrypting the ith packet. Let C_2 be the first block of the ith ciphertext. And let C_3 be the last CBC block of the underlying ciphertext the user just output (i.e., the user will use C_3 as its next IV). If the adversary is able to force the user to encrypt the block $C_1 \oplus C_3 \oplus G$, where $\oplus$ is the XOR operation, and if the resulting block is C_2, the adversary knows its guess of for G was correct; otherwise the adversary knows its guess was incorrect.

A small complication arises when mounting this attack against SSH-IPC because the attacker cannot control the entire first block of an encoded message (because the first 40 bits of an encoded packet contain metadata). This means that an attacker may not be able to force a user's underlying CBC scheme to encrypt the block $C_1 \oplus C_3 \oplus G$. An attacker will, however, be able to mount this attack if C_1 and C_3 are identical in the bits that the attacker cannot control. Let l be the block length (in bits) of the underlying block cipher. Since an attacker can control approximately $\lg(l/8)$ bits of the padding length field and approximately $15 - \lg(l/8)$ bits of the packet length field of an encoded message (SSH implementations are only required to support packets with payloads containing less than 2^{15} bytes and all packets must be padded to a multiple of the block length), an attacker could mount a variant of the above attack by waiting for a collision on approximately 25 bits (but the adversary's last encryption request may be up to 2^{15} bytes long).

4. Attacks Against a Natural "Fix"

The problem with SSH-IPC stems from the fact that its underlying encryption scheme is itself vulnerable to chosen-plaintext attacks. A logical fix might therefore be to replace the underlying encryption scheme with randomized CBC mode (i.e., CBC mode in which a new random IV is chosen for each message; this new IV must also be sent with the ciphertext). Randomized CBC mode was proven to resist chosen-plaintext attacks in [2]. We refer to an SSH implementation that uses randomized CBC mode as SSH-NPC, or SSH with no packet chaining.

It is possible to prove that SSH-NPC preserves privacy against chosen-plaintext attacks and integrity under a notion called "integrity of plaintexts" provided that a user does not use SSH-NPC to encrypt more than 2^{32} messages with any given key. This proof holds even if the paddings used in encoded packets are not random (which is allowed by the SSH specification). As the following attack shows, however, even though SSH-NPC with non-random padding preserves privacy against chosen-plaintexts attacks, it does not preserve privacy against chosen-ciphertext attacks.

REACTION ATTACK AGAINST SSH-NPC. The SSH specification encourages, although does not require, implementations to use random padding. Unfortunately, when the padding value is fixed (e.g., all zeros), SSH-NPC is susceptible to an easily-mountable reaction attack. Furthermore, this attack can be made to work even when the padding values are not fixed but short and not hard to predict: an attacker can simply wait until the predicted padding values collide

and then use the predicted value to successfully mount an attack. The attack we describe here is similar in spirit to Wagner's attack in [7] and to the attacks in [20, 26] (the term "reaction attack" comes from [16]).

The attack proceeds roughly as follows: an attacker intercepts (and prevents the delivery of) two ciphertexts sent by one party involved in an SSH connection. The adversary then makes a guess about the relationship between the two plaintexts corresponding to the two intercepted ciphertexts. The adversary then uses that guess and those two ciphertexts to create a new "ciphertext," which the adversary then sends to the other party involved in the SSH session. By observing the second party's reaction (recall that if the second party does not accept the doctored ciphertext, the connection will be terminated), an adversary will learn whether its guess was correct. Intuitively, this attack works because an attacker can modify the ciphertext in such a way that if its guess was correct, the ciphertext that the second party receives will verify. If its guess was incorrect, with high probability the ciphertext will not verify.

We now describe the attack in more detail. As before, let $\oplus$ denote the XOR operation, let $\|$ denote the concatenation of two strings, and let l denote the block length (in bits) of the block cipher that SSH-NPC uses in CBC mode. Suppose a user uses SSH-NPC to encrypt two equal-length messages M_1 and M_2 with lengths at most $l - 40$ (or that are identical after their $l - 40$-th bit). (For simplicity of exposition, assume that the two messages are exactly $l - 40$ bits long.) Let P_{11} and P_{12} be the first and the second block of the encoded packet corresponding to the payload M_1, respectively. Similarly, let P_{21} and P_{22} be the first and the second block of the encoded packets corresponding to M_2, respectively. The blocks P_{11} and P_{21} correspond to the packet length, the padding length, and the payload fields of the two encoded packets, and the blocks P_{12} and P_{22} correspond to the padding fields. Since we are assuming fixed padding (such as padding with all zeros), the padding blocks P_{12} and P_{22} will be equal.

When SSH-NPC's underlying CBC mode encryption scheme encrypts the first encoded packet $P_{11}\|P_{12}$, it will generate a ciphertext $\sigma_1 = C_{10}\|C_{11}\|C_{12}$; SSH-NPC's underlying MAC will also generate a tag τ_1 (the MAC being computed over the concatenation of a counter and $P_{11}\|P_{12}$). Similarly, SSH-NPC will generate the CBC ciphertext $C_{20}\|C_{21}\|C_{22}$ and the MAC tag τ_2 for the encoded packet $P_{21}\|P_{22}$. (The two blocks C_{10} and C_{20} correspond to the underlying CBC mode's random initialization vectors.)

Now assume that the receiver has not yet received the two ciphertexts corresponding to M_1 and M_2 (i.e., the recipient's counter is identical to the counter that the sender used when she encrypted the first message). Assume that the attacker knows either M_1 or M_2 and wants to verify a guess of the other (or that the attacker wants to verify a guess of the relationship between M_1 and M_2). Let X be the value $P_{11} \oplus P_{21} \oplus C_{20}$ (recall that the blocks P_{11} and P_{12} both begin with the same 40 bits of header information and that they respectively end in M_1 and M_2). The attacker then asks the receiver to decrypt the message $X\|C_{21}\|C_{22}\|\tau_1$. If the attacker's guess is correct, then $X\|C_{21}\|C_{22}$ will decrypt, via SSH-NPC's underlying CBC scheme, to $P_{11}\|P_{12}$, the MAC tag τ_1 will verify, and the decryptor will accept the message. If the attacker's guess is incorrect, however, $X\|C_{21}\|C_{22}$ will not decrypt to $P_{11}\|P_{12}$, the tag τ_1 will not verify (unless the attacker also succeeds in breaking the security of the underlying MAC scheme), and the SSH-NPC connection will terminate. The adversary, by watching the recipients reaction, therefore learns information about the plaintexts the sender is encrypting.

There are two aspects of this attack that make it easy to mount. First, this attack only requires modifying encrypted packets; no chosen-plaintexts are required. Second, an attacker can learn whether its guess is correct simply by watching the recipient's response. These observations mean that all an attacker needs to perform this attack is the ability to monitor, prevent the delivery of, and inject messages in the encrypted communications between two parties. Similar to Wagner's attack in [7], this attack can be used to (among other things) infer the characters that a user types over an interactive SSH-NPC session. Of course, once the attacker makes an incorrect guess, SSH-NPC terminates the connection. Nonetheless, an attacker might still be able to repeat its attack after the user begins a new session.

INFORMATION LEAKAGE, REPLAY, AND OUT-OF-ORDER DELIVERY ATTACKS. Although the SSH draft suggests that an SSH session rekey after every gigabyte of transmitted data, doing so is not required. We caution that if an SSH-NPC (or SSH-IPC) session is not rekeyed frequently enough, then the session will be vulnerable to a number of other attacks. Recall that the SSH binary packet protocol includes a 32-bit counter in each message to be MACed. These attacks make use of the fact that if the SSH connection is not rekeyed frequently enough, then the counter will begin to repeat.

Recall that SSH generates each MAC using the encoded payload prepended with a counter as an input and then appends the MAC to the intermediate ciphertext to generate a ciphertext packet. As a result, if the underlying MAC algorithm is stateless and deterministic (which many MACs are), then allowing the counter to repeat will leak information about a user's plaintexts (through the MAC). We present the attacks in more details for completeness. Suppose that the underlying message authentication scheme is stateless and deterministic and that the padding is some fixed value. Suppose that an attacker A sees a ciphertext with a MAC tag τ and suspects that the underlying payload is M. To verify its guess, A waits for the sender to encrypt $2^{32} - 1$ more packets and then requests the sender to encrypt the payload M. Let τ' be the MAC tag returned in response to the request. If A's guess is correct, then τ' will equal τ. Otherwise $\tau' \neq \tau$ with very high probability. The attack can also be used to break the privacy of SSH-NPC when SSH-NPC uses random padding. In particular, if the first 2^{32} messages that a user tags result in encoded packets that use the minimum 4 bytes of random padding, then an attacker capable of forcing a user to tag an additional 2^{32} chosen-plaintexts will be able to learn information about the user's initial 2^{32} messages. The property used in this attack (that tagging with a deterministic MAC leaks information about plaintexts) was also exploited by [5] and [20].

If the counter is allowed to repeat, SSH-NPC also becomes vulnerable to replay attacks and out-of-order delivery attacks. For replay attacks, once the receiver has decrypted 2^{32} messages, an attacker will be able to convince the receiver to re-accept a previously received message. For out-of-order delivery attacks, after the sender has encrypted more that 2^{32} messages, an attacker will be able to modify the order in which the messages are decrypted.

5. Secure Fixes to SSH

We now briefly describe our new SSH instantiations. We show in Section 6 that these new alternatives provably meet our strongest notions of security. That is, assuming that these fixes are not used to encrypt more than 2^{32} packets between rekeying, these new constructions will resist chosen-plaintext and chosen-ciphertext privacy attacks as well as forgery, replay, and out-of-order delivery attacks. Security above 2^{32} is not guaranteed because, after 2^{32} packets are encrypted, the SSH BPP's 32-bit internal counter will begin to wrap. We will compare these instantiations of SSH to others and discuss additional possible modifications, including extending the length of SSH's internal counter, in Section 7.

SSH VIA RANDOMIZED CBC MODE WITH RANDOM PADDING: SSH-$NPC. Recall that the attack against SSH-NPC involves creating a new intermediate ciphertext that would decrypt to an encoded packet that the user previously encrypted (assuming the attacker's guess was correct). With this in mind, we propose a provably secure SSH instantiation (SSH-$NPC) that uses randomized CBC mode for the underlying encryption scheme and that requires that encoded packets use random padding. We require that the random padding be chosen anew for each encryption and that the random padding occupy at least one full block of the encoded packet. This conforms to the current SSH specification since the latter allows padding up to 255 bytes.

The intuition behind the security of this alternative and the reason that this alternative resists the attack in Section 4 is the following. Since the random padding is not sent in the clear, an attacker will not know what the random padding is and will not be able to forge a ciphertext that will decrypt to that previously encoded message (with the same random padding). Furthermore, any other attack against SSH-$NPC would translate into an attack against the underlying CBC mode encryption scheme, the underlying MAC, the encoding scheme, or the underlying block cipher.

SSH VIA CBC MODE WITH CTR GENERATED IVS: SSH-CTRIV-CBC. Instead of using CBC mode with a random IV, it is also possible to generate a "random-looking" IV by encrypting a counter with a different key; we call this alternative SSH-CTRIV-CBC. Unlike SSH-$NPC, for SSH-CTRIV-CBC we do *not* require a full block of padding and we do not require the padding to be random. The reason we do not require random padding for this alternative is because the decryptor is stateful and that any modification to an underlying CBC ciphertext will, with probability 1, change the encoded packet. This alternative is more attractive than SSH-$NPC because it does not increase the size of ciphertexts compared to SSH-IPC (but it does require one additional block cipher application compared to SSH-IPC).

SSH VIA CTR MODE WITH STATEFUL DECRYPTION: SSH-CTR. SSH-CTR uses standard CTR mode as the underlying encryption scheme with one modification: both the sender and the receiver maintain the counters themselves, rather than transmitting them as part of the ciphertexts. We refer to this variant of CTR mode as *CTR mode with stateful decryption*. We point out that this CTR mode variant offers the same level of chosen-plaintext privacy as standard CTR mode, the security of which was shown in [2]. As with SSH-CTRIV-CBC, SSH-CTR does not require additional padding and does not require the padding to be random. Furthermore, unlike SSH-$NPC and SSH-CTRIV-CBC, SSH-CTR requires the same number of block cipher invocations as SSH-IPC.

OTHER POSSIBILITIES. There are numerous other possible fixes to the SSH BPP. Rather than enumerate all possible fixes to the SSH BPP, in Section 6 we discuss how one can use our general proof techniques to prove the security of other fixes (assuming, of course, that the other fixes are indeed secure). For example, another fix of interest might be SSH-EIV-CBC, or SSH where the underlying encryption scheme is replaced by a CBC variant in which the IV is the *encipherment* of the last block of the previous ciphertext.

6. Provable Security Results

PRACTICE-ORIENTED PROVABLE SECURITY. Provable security was first introduced by Goldwasser and Micali in [15] and has since gained wide acceptance in both theoretical and applied cryptography. In this approach, one determines the security definitions and adversary models for the cryptographic construct in question and "proves" security of the desired construct via a *reduction* from the hardness of the underlying primitives. We follow this approach here. Additionally, our reductions allow concrete bounds to be readily obtained, thus following the *practice-oriented* approach to provable security. We note, however, that for simplicity we do not state exact bounds in the paper but simply state roughly how the resources for breaking the construct and those for breaking the underlying primitives compare. In most cases, they are equal. In other cases, they can be easily determined by looking at the expansion between a payload message and its encoded packet.

ANALYZING SSH VIA A NEW PARADIGM. An SSH ciphertext is the concatenation of the encryption and the MAC of (some encodings of) an underlying payload message. At first glance, this seems to fall into the "Encrypt-and-MAC" method of composing an encryption scheme with a MAC scheme: to encrypt a message M, apply the encryption algorithm to M and the tag generation algorithm to M, then concatenate the resulting strings to produce the final ciphertext to be transmitted. As pointed out in [5, 20], this particular composition method is *not* generically secure: security under standard notions of the encryption and MAC schemes used as building blocks under this composition method is not enough to guarantee the privacy of the payload. Naturally, this raises a question regarding the security of SSH.

We show here that, with an appropriate encoding method, such as the method used in SSH, an Encode-then-E&M scheme can actually be made secure. In fact, our analysis models SSH more generally as an authenticated encryption scheme constructed via a paradigm we call *Encode-then-E&M*: to encrypt a message, first encode it (as SSH does), then encrypt and MAC the encoded packets. Our analysis was done in a general way in order to ensure that that the definitions and techniques we developed will be useful to the evaluators of other SSH-like schemes.

As described in Section 2, an SSH BPP encoded message (for encryption) consists of a packet length field, a padding length field, payload data, and padding. An encoded message (for MACing) is identical to an encoded message for encryption except that it is prepended with a 32-bit counter.

6.1 Definitions

NOTATION. If x and y are strings, then $|x|$ denotes the length of x in bits and $x\|y$ denotes their concatenation. If i is a non-negative integer, then $\langle i \rangle_l$ denotes the unsigned l-bit binary representation of i. The empty string is denoted ε. When we say an algorithm is stateful, we mean that it uses and updates its state and that the entity executing it maintains the state between invocations. Let ε denote the initial state of any (stateful or stateless) algorithm. If f is a randomized (resp., deterministic) algorithm, then $x \xleftarrow{R} f(y)$ (resp., $x \leftarrow f(y)$) denotes the process of running f on input y and assigning the result to x. If A is a program, $A \Leftarrow x$ means "return the value x to A."

ENCRYPTION SCHEMES WITH STATEFUL DECRYPTION. As usual a *symmetric encryption scheme* or *authenticated encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms. The randomized key generation algorithm returns a key K. The encryption algorithm, which may be both randomized and stateful, takes key K and a plaintext and returns a ciphertext. Motivated by SSH, the novel feature here is that the decryption algorithm may also be stateful (but not randomized); the decryption algorithm takes key K and a ciphertext and returns either a plaintext or a special symbol $\perp$ (indicating failure).

Consider the interaction between an encryptor and a decryptor. If, at any point in time, the sequence of inputs to the decryptor is not a prefix of the sequence of outputs of the encryptor, then we say that the encryption and decryption processes have become *out-of-sync* and refer to the decryption input at that point in time as the first *out-of-sync* input. The usual correctness condition, which said that if C is produced by encrypting M under K then decrypting C under K yields M, is replaced with a less stringent condition requiring only that decryption succeed when the encryption and decryption processes are in-sync. More precisely, the following must be true for any key K and plaintexts $M_1, M_2, \ldots$. Suppose that both $\mathcal{E}_K$ and $\mathcal{D}_K$ are in their initial states. For $i = 1, 2, \ldots$, let $C_i = \mathcal{E}_K(M_i)$ and let $M'_i = \mathcal{D}_K(C_i)$. It must be that $M_i = M'_i$ for all i.

MESSAGE AUTHENTICATION SCHEMES. A *message authentication scheme* or *MAC* $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms. The randomized key generation algorithm returns a key K. The tagging algorithm, which may be both randomized and stateful, takes key K and a plaintext and returns a tag. The deterministic and stateless verification algorithm takes key K, a plaintext, and a candidate tag and returns a bit. For any key K and message M, and for any internal state of $\mathcal{T}_K$, we require that $\mathcal{V}_K(M, \mathcal{T}_K(M)) = 1$.

ENCODING SCHEMES. An "encoding" is an *unkeyed* transformation. We use encodings to model the process of loading a payload message into a packet for encryption and a packet for message authentication (recall that the encoded packet that the SSH BPP encrypts is slightly different than the encoded packet that the SSH BPP MACs). Syntactically, an *encoding scheme* $\mathcal{EC} = (Enc, Dec)$ consists of an encoding algorithm and a decoding algorithm. The encoding algorithm Enc, which may be both randomized and stateful, takes as input a message M and returns a pair of messages (M_e, M_t). The decoding algorithm Dec, which may also be stateful but not randomized, takes as input a message M_e and returns a pair of messages (M, M_t), or $(\perp, \perp)$ on error. The following consistency requirement must be met. Consider any two messages M, M' where $|M| = |M'|$. Let $(M_e, M_t) \xleftarrow{R} Enc(M)$ for Enc in some state, and let $(M'_e, M'_t) \xleftarrow{R} Enc(M')$ for Enc is in some (possibly different) state. We require that $|M_e| = |M'_e|$ and $|M_t| = |M'_t|$. Furthermore, suppose that both Enc and Dec are in their initial states. For any sequence of messages $M^1, M^2, \ldots$ and for $i = 1, 2, \ldots$, let $(M^i_e, M^i_t) = Enc(M^i)$, and then let $(m^i, m^i_t) = Dec(M^i_e)$. We require that $M^i = m^i$ and that $M^i_t = m^i_t$ for all i.

ENCODE-THEN-E&M PARADIGM. Now consider an encoding scheme, and let (M_e, M_t) be the encoding of some message M. To generate a ciphertext for M using the Encode-then-E&M construction, the message M_e is encrypted with an underlying encryption scheme, the message M_t is MACed with an underlying MAC algorithm, and the resulting two values (intermediate ciphertext and MAC) are concatenated to produce the final ciphertext. The composite decryption procedure is similar except the way errors (e.g., decoding problems or tag verification failures) are handled: in particular, should the composite decryption algorithm enter a new state or return to its previous state? We take the approach used in SSH whereby, if a decryption fails, the composite decryption algorithm enters a "halting state." This approach is perhaps the most intuitive since, upon detecting a chosen-ciphertext attack, the decryption algorithm prevents all subsequent ciphertexts from being decrypted (of course, this also makes the decryptor vulnerable to a denial-of-service-type attack). Construction 1 shows the Encode-then-E&M composition method in details.

Construction 1. **(Encode-then-E&M)** Let $\mathcal{EC} = (Enc, Dec)$, $\mathcal{SE} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$, and $\mathcal{MA} = (\mathcal{K}_t, \mathcal{T}, \mathcal{V})$ be encoding, encryption, and message authentication schemes with compatible message spaces (the outputs from Enc are suitable inputs to $\mathcal{E}$ and $\mathcal{T}$). Let all states initially be ε. We associate to these schemes a composite *Encode-then-E&M scheme* $\overline{\mathcal{SE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ as follows:

Algorithm $\overline{\mathcal{K}}$	Algorithm $\overline{\mathcal{E}}_{\langle K_e, K_t \rangle}(M)$
$K_e \xleftarrow{R} \mathcal{K}_e$ $K_t \xleftarrow{R} \mathcal{K}_t$ Return $\langle K_e, K_t \rangle$	$(M_e, M_t) \xleftarrow{R} Enc(M)$ $\sigma \xleftarrow{R} \mathcal{E}_{K_e}(M_e)$; $\tau \xleftarrow{R} \mathcal{T}_{K_t}(M_t)$ $C \leftarrow \sigma \| \tau$ Return C

Algorithm $\overline{\mathcal{D}}_{\langle K_e, K_t \rangle}(C)$
If $st = \perp$ then return $\perp$
If cannot parse C then $st \leftarrow \perp$; return $\perp$
Parse C as $\sigma \| \tau$; $M_e \leftarrow \mathcal{D}_{K_e}(\sigma)$
If $M_e = \perp$ then $st \leftarrow \perp$; return $\perp$
$(M, M_t) \leftarrow Dec(M_e)$
If $M = \perp$ then $st \leftarrow \perp$; return $\perp$
$v \leftarrow \mathcal{V}_{K_t}(M_t, \tau)$
If $v = 0$ then $st \leftarrow \perp$; return $\perp$
Return M

Although only $\overline{\mathcal{D}}$ explicitly maintains state in the above pseudocode, the underlying encoding, encryption, and MAC schemes may also maintain state. ∎

6.2 Security Notions

Since the goal is to model schemes based on block ciphers and cryptographic hash functions, a concrete security

treatment is used. We associate to any adversary a number called its "advantage" that measures its success in breaking a given scheme with respect to a given security notion. The smaller an adversary's advantage is against a given scheme, the stronger that scheme is with respect to that adversary. In discussion, take "secure" to mean that the advantage of any adversary with "practical" resources is "small." We briefly describe the security notions here. The full version of this paper [4] presents these notions in more detail.

SECURITY NOTIONS FOR ENCRYPTION SCHEMES WITH STATEFUL DECRYPTION. A secure authenticated encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is one that preserves both privacy and integrity. The standard notion of indistinguishability (privacy) under chosen-plaintext attacks (IND-CPA) is as follows [2]: we consider a game in which an adversary A is given access to an *left-or-right-encryption* (lr-encryption) oracle $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, for some hidden bit b, that on input two equal length message M_0, M_1, returns $\mathcal{E}_K(M_b)$. After performing a number of lr-encryption queries, the adversary must return a guess for the bit b. We define $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(A)$ as the probability that A returns 1 when $b = 1$ minus the probability that A returns 1 when $b = 0$.

For our notion of chosen-ciphertext privacy for stateful decryption (IND-SFCCA), we consider a game in which an adversary B is given access to an lr-encryption oracle $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$ and a decryption oracle $\mathcal{D}_K(\cdot)$. As long as B's queries to $\mathcal{D}_K(\cdot)$ are in-sync with the responses from $\mathcal{E}_K(\mathcal{LR}(\cdot,\cdot,b))$, the decryption oracle performs the decryption (and updates its internal state) but does not return a response to B. Once B makes an out-of-sync query to $\mathcal{D}_K(\cdot)$, the decryption oracle returns the output of the decryption. We define $\mathbf{Adv}^{\text{ind-sfcca}}_{\mathcal{SE}}(B)$ as the probability that B returns 1 when $b = 1$ minus the probability that B returns 1 when $b = 0$. The new ind-sfcca notion implies the previous notion of indistinguishability under chosen-ciphertext attacks (IND-CCA [2]). Note that, without allowing an adversary to query the decryption oracle with in-sync ciphertexts (e.g., in the standard ind-cca setting), we would not be able to model attacks in which the adversary attacks a stateful decryptor after the latter had decrypted a number of legitimate ciphertexts (perhaps because of some weakness related to the state of the decryptor at that time).

The standard notion for integrity of plaintexts (INT-PTXT) is as follows [5]: we consider a game in which an adversary A is given access to an encryption oracle $\mathcal{E}_K(\cdot)$ and a decryption-verification oracle $\mathcal{D}^*_K(\cdot)$. On input a candidate ciphertext C, the decryption-verification oracle invokes $\mathcal{D}_K(C)$ and returns 1 if $\mathcal{D}_K(C) \neq \perp$ and 0 otherwise. We define $\mathbf{Adv}^{\text{int-ptxt}}_{\mathcal{SE}}(A)$ as the probability that A can find a ciphertext C such that $\mathcal{D}^*_K(C) = 1$ but that the decrypted value of C, i.e. $\mathcal{D}_K(C)$, was not previously a query to $\mathcal{E}_K(\cdot)$. For our notion of integrity of ciphertexts for stateful decryption (INT-SFCTXT), we again consider a game in which an adversary B is given access to the two oracles $\mathcal{E}_K(\cdot)$ and $\mathcal{D}^*_K(\cdot)$. We define $\mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{SE}}(B)$ as the probability that B can generate a ciphertext C such that $\mathcal{D}^*_K(C) = 1$ and C is an out-of-sync query. The new notion of int-sfctxt implies the previous notion of integrity of ciphertexts (INT-CTXT [5]) as well as security against replay and out-of-order delivery attacks.

The following proposition states that, if a scheme is indistinguishable under chosen-plaintexts attacks and if the scheme meets our strong definition of integrity of ciphertexts, then the scheme will meet our strong definition of indistinguishability under chosen-ciphertext attacks. The proof appears in [4]. It is similar to the results in [5] and [19] which show that the standard ind-cpa and the standard int-ctxt notion imply the standard ind-cca notion.

PROPOSITION 1. *Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric authenticated encryption scheme. Given any ind-sfcca adversary A, we can construct an int-sfctxt adversary I and an ind-cpa adversary B such that*

$$\mathbf{Adv}^{\textit{ind-sfcca}}_{\mathcal{SE}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{int-sfctxt}}_{\mathcal{SE}}(I) + \mathbf{Adv}^{\textit{ind-cpa}}_{\mathcal{SE}}(B)$$

and I and B use the same resources as A. ∎

UNFORGEABILITY OF MAC SCHEMES. We consider a secure MAC $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ to be one that is strongly unforgeable under chosen-message attacks [5]. We consider a game in which a forger F is given access to a tagging oracle $\mathcal{T}_K(\cdot)$ and a verification oracle $\mathcal{V}_K(\cdot)$. The forger is allowed arbitrary queries to the oracles and wins if it can find a pair (M, τ) such that $\mathcal{V}_K(M, \tau) = 1$ but τ was never returned by $\mathcal{T}_K(\cdot)$ as a tag for M. We denote the advantage of this forger as $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}}(F)$. Although this notion is in general stronger than the standard notion of unforgeability [3], we note that any pseudorandom function is a strongly unforgeable MAC, and most practical MACs seem to be strongly unforgeable.

PSEUDORANDOM FUNCTIONS. We formalize pseudorandom functions and their security following [14, 3]. Suppose $\mathcal{F}$ is a family of functions from some message space $\mathcal{M}$ to $\{0,1\}^L$, and let $\text{Rand}^{\mathcal{M}\to L}$ denote the family of all functions from $\mathcal{M}$ to $\{0,1\}^L$. We define $\mathbf{Adv}^{\text{prf}}_{\mathcal{F}}(D)$ as the advantage of a distinguisher D in distinguishing a random instance of $\mathcal{F}$ from a random instance of $\text{Rand}^{\mathcal{M}\to L}$.

COLLISION RESISTANCE OF ENCODING SCHEMES. The security of a composite Encode-then-E&M construction depends on properties of the underlying encoding, encryption, and MAC schemes. In addition to the standard assumptions of indistinguishability of the encryption scheme and unforgeability and pseudorandomness of the MAC scheme, we require "collision resistance" of the encoding scheme. We motivate this notion as follows. Consider an integrity adversary against a composite Encode-then-E&M scheme. If the adversary can find two different messages that encode (or decode) to the same input for the underlying MAC, then the adversary may be able to compromise the integrity of the composite scheme. Consider now an indistinguishability adversary against the composite scheme. As long as the adversary does not generate two inputs for the underlying MAC that collide, the underlying MAC should not leak information about the plaintext. The following describes the notions of collision resistance for encoding schemes. Formal definitions appear in [4].

An adversary A mounting a "chosen-plaintext attack" against an encoding scheme $\mathcal{EC} = (Enc, Dec)$ is given access to an encoding oracle $Enc(\cdot)$. If A can make the encoding oracle output two pairs that collide on their second components (i.e., the M_t's), then A wins. We allow A to repeatedly query the encoding oracle with the same input. Similarly, an adversary B mounting a "chosen-ciphertext attack" against $\mathcal{EC}$ is given access to both an encoding oracle and a decoding oracle $Dec(\cdot)$. If B can cause a collision in the second components of the outputs of $Enc(\cdot)$, $Dec(\cdot)$, or both, then

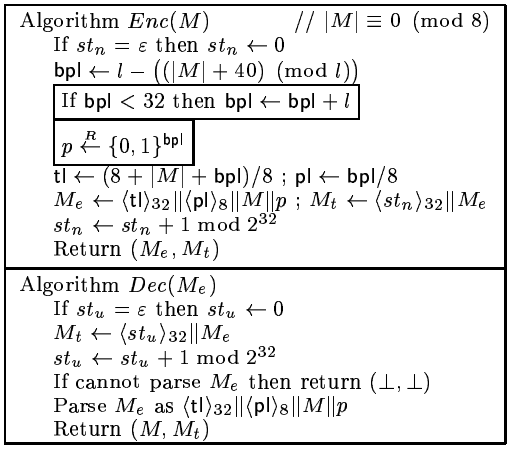

Figure 2: The SSH encoding algorithm $\mathcal{EC} = (Enc, Dec)$ for l-bit blocks, where $l \equiv 0 \pmod 8$ and $64 \leq l \leq 252 \cdot 8$. The states st_n and st_u are maintained across invocations.

it wins. Of course, we exclude the cases where B uses the two oracles in a trivial way to obtain collisions (e.g. submitting a query to $Enc(\cdot)$ and then immediately submitting the first component of the result, namely M_e, to $Dec(\cdot)$). We refer to the advantages of the adversaries in these two settings as $\mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A)$ and $\mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(B)$, respectively. All encoding schemes with deterministic and stateless encoding algorithms are insecure under chosen-plaintext collision attacks. Furthermore, all encoding schemes with stateless decoding algorithms are insecure under chosen-ciphertext collision attacks.

6.3 SSH Security Results

The SSH encoding scheme, when used with an l-bit block cipher, is shown in Figure 2 (see also Section 2). Recall that $|x|$ denotes the length of string x in bits, not bytes, and that $\langle x \rangle_k$ denotes the representation of x as a k-bit unsigned integer. As mentioned, although Figure 2 shows the padding p as a random string (the second boxed equation), the SSH specification does not require that p be random. Additionally, although the SSH specification allows up to 255 bytes of padding, the two major implementations of the SSH protocol, `openssh-2.9p2` and SSH Communications' `ssh-3.0.1`, use the minimum-recommended padding length shown in Figure 2. The proposed SSH-$NPC instantiation of SSH replaces the first boxed statement with $\mathsf{bpl} \leftarrow \mathsf{bpl} + l$ if $\mathsf{bpl} < l$ and *always* uses random padding as shown in the second boxed statement. The instantiations SSH-CTRIV-CBC, SSH-EIV-CBC, and SSH-CTR, on the other hand, uses the first boxed statement with no modification and allows padding p to be non-random.

The following lemma gives the collision bounds for the SSH encoding as shown in Figure 2. Notice that if $q_e \leq 2^{32}$, then $\lceil q_e \cdot 2^{-32} \rceil - 1 \leq 0$ and $\mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A) = 0$ for any adversary A. Also, if a coll-cca adversary C submits more than 2^{32} encoding queries or 2^{32} decoding queries, then it can completely break the scheme, i.e. $\mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) = 1$. (For coll-cca security of up to 2^{32} decoding queries it is critical that the decoding algorithm increment its counter on every invocation, even for messages that do not correctly decode.) The proof appears in [4].

LEMMA 1. ***(Collision Resistance of the SSH Encoding)*** *Let $\mathcal{EC}$ be the encoding scheme shown in Figure 2 and let mbpl be the minimum padding length (32 bits in Figure 2; the 32 in the equations below corresponds to the length of the encoding scheme's internal counter, not the minimum padding length). For any coll-cpa adversary A and any coll-cca adversary B, each making q_e encoding queries and, in the case of B, making q_d decoding queries, we have that*

$$\begin{aligned} \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(A) &\leq \lceil q_e \cdot 2^{-32} \rceil \cdot \left(\lceil q_e \cdot 2^{-32} \rceil - 1 \right) \cdot 2^{31 - \mathsf{mbpl}} \\ \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(B) &= 0 \quad \text{if } q_e, q_d \leq 2^{32} \end{aligned}$$

and that coll-cca collision resistance is not provided if q_e or $q_d > 2^{32}$. ∎

INTEGRITY AND PRIVACY OF OUR RECOMMENDATIONS. Our proposed fixes from Section 5 are secure under our strong notions of integrity (int-sfctxt) and indistinguishability (ind-sfcca). We sketch our proof of security for SSH-CTR (see [4] for details). The proof technique extends naturally to other possible fixes to the SSH BPP.

We first present a general result that holds for all Encode-then-E&M constructions. This result states that, for any Encode-then-E&M construction, if the underlying encryption scheme is ind-cpa-secure, if the underlying MAC is a secure pseudorandom function, and if the encoding scheme is coll-cpa collision resistant, then the composite Encode-then-E&M scheme will be ind-cpa-secure. The proof in [4] uses a slightly weaker notion of security for MACs.

LEMMA 2. ***(Privacy of Encode-then-E&M with Respect to Chosen-Plaintext Attacks)*** *Let $\mathcal{SE}$, $\mathcal{MA}$, and $\mathcal{EC}$ respectively be an encryption, a message authentication, and an encoding scheme. Let $\overline{\mathcal{SE}}$ be the encryption scheme associated to them as per Construction 1. Then, given any ind-cpa adversary S against $\overline{\mathcal{SE}}$, we can construct adversaries A, D, and C such that*

$$\begin{aligned} \mathbf{Adv}^{ind\text{-}cpa}_{\overline{\mathcal{SE}}}(S) \leq\ & \mathbf{Adv}^{ind\text{-}cpa}_{\mathcal{SE}}(A) + 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(D) + \\ & 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(C) \,. \end{aligned}$$

Furthermore, A, D, and C use the same resources as S except that A's and D's inputs to their respective oracles may be of different lengths than those of S (due to the encoding). ∎

We now state our result for SSH-CTR:

THEOREM 1. ***(Security of* SSH-CTR*)*** *Let $\mathcal{SE}$ be a CTR-mode encryption scheme with stateful decryption, let $\mathcal{MA}$ be a message authentication scheme, and let $\mathcal{EC}$ be the encoding scheme described above. Let* SSH-CTR *be the encryption scheme associated to them as per Construction 1. Then, given any int-sfctxt adversary I against* SSH-CTR*, we can construct adversaries F and C such that Equation (1) holds. Similarly, given any ind-sfcca adversary A against* SSH-CTR*, we can construct adversaries S, B, E, and G such that Equation (2) holds.*

$$\mathbf{Adv}^{\text{int-sfctxt}}_{\text{SSH-CTR}}(I) \leq \mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}}(F) + \mathbf{Adv}^{\text{coll-cca}}_{\mathcal{EC}}(C) \qquad (1)$$

$$\begin{aligned} \mathbf{Adv}^{ind\text{-}sfcca}_{\text{SSH-CTR}}(A) \leq\ & 2 \cdot \mathbf{Adv}^{\text{int-sfctxt}}_{\text{SSH-CTR}}(S) + \mathbf{Adv}^{ind\text{-}cpa}_{\mathcal{SE}}(B) + \\ & 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathcal{MA}}(E) + 2 \cdot \mathbf{Adv}^{\text{coll-cpa}}_{\mathcal{EC}}(G) \qquad (2) \end{aligned}$$

Furthermore, F and C use the same resources as I except that F's messages to its oracles may be of different lengths than I's queries to its oracles (due to encoding) and C's messages to its decoding oracle may have slightly different lengths than I's decryption queries. Also, S, B, E, and G use the same resources as A except that B's and E's inputs to their respective oracles may be of different lengths than those of A (due to the encoding). ∎

Theorem 1 can be interpreted as follows. Equation (1) states that SSH-CTR provides stateful chosen-ciphertext integrity if the MAC is strongly unforgeable and if the encoding is coll-cca collision resistant. Equation (2) states that SSH-CTR provides stateful chosen-ciphertext privacy if it provides stateful chosen-ciphertext integrity, if the underlying encryption scheme is ind-cpa secure, if the MAC is a secure pseudorandom function, and if the encoding is coll-cpa secure. As as example, making reasonable assumptions about the security of the HMAC scheme, an implementation of SSH-CTR that uses HMAC and AES in stateful-decryption CTR mode will be secure under both of the strong notions provided that at most 2^{32} messages are encrypted between rekeying. Notice here that we use different security properties of the MAC to obtain different security aspects of SSH-CTR, namely strong unforgeability for integrity and pseudorandomness for privacy. This distills the property of the MAC that leads to each aspect of security. We point out, however, that the notion of strong unforgeability is relatively new [5] and that we do not know of any provably-secure strongly unforgeable MACs that are not also pseudorandom functions.

To prove Theorem 1 (details in [4]), we first use Lemma 1, Lemma 2, the ind-cpa proof of security for CTR mode [2], and the assumed pseudorandomness of the underlying MAC to show that SSH-CTR is ind-cpa-secure. We then prove Equation (1). Applying Proposition 1 and our ind-cpa and int-sfctxt results for SSH-CTR leads to Equation (2). We briefly discuss our proof of Equation (1). Let I be an int-sfctxt adversary and let M^i be I's i-th chosen-plaintext query to its encoding oracle, let M_e^i, M_t^i be the encoding of M^i, and let $\sigma_i \| \tau_i$ be the returned ciphertext. Let $\sigma'_j \| \tau'_j$ be I's j-th decryption-verification oracle query, let m_e^j be the decryption of σ'_j by the underlying decryption algorithm. To prove Equation (1), we basically show that given an int-sfctxt adversary attacking SSH-CTR, that adversary can also be used to attack the unforgeability of the underlying MAC, to attack the coll-cca collision resistance of the underlying encoding scheme, or that the first out-of-order ciphertext submitted by the adversary, $\sigma'_j \| \tau'_j$, is such that $\sigma_j \neq \sigma'_j$ but $M_e^j = m_e^j$. By properties of CTR mode with stateful decryption, the latter event cannot occur. The same property holds for SSH-CTRIV-CBC and SSH-EIV-CBC. For SSH-$NPC the latter event can occur, but the probability the latter event occurs is small because the last (random) block of the encoded packet is not given to the adversary. The strategy we outlined in this paragraph can be used to prove the security of other fixes to the SSH BPP that work by replacing the underlying encryption scheme; namely, prove that the underlying encryption scheme is ind-cpa secure and that the probability of the event we described is small. (We only consider the first out-of-order ciphertext query an adversary makes because if the first out-of-order ciphertext query does not decrypt, the decryptor enters a halting state.)

7. Discussion and Recommendations

Having thus presented our main results, we are now in a position to make specific recommendations to the SSH community. We begin by noting that a fundamental problem with the current SSH specification is that the counter (that is prepended to the encoded payload before MACing) is only 32 bits long. As shown in Section 4, once the 32 bit counter repeats, an SSH session's MAC tags may begin to leak information about a user's plaintexts. Our provable security results reflect this constraint: strong security is maintained only if the parties rekey at least once every 2^{32} packets. Two natural solutions to this problem are to either make the counter longer or to *require* an SSH session to rekey at least once every 2^{32} messages. We recommend the second option because it does not affect the packet format and thus will likely require minimal changes to existing implementations of SSH. In the following discussion we assume that all implementations will rekey frequently.

We consider the current instantiation of the SSH BPP transport protocol, SSH-IPC, and our specific recommendations. We also consider two other possible alternatives, namely switching to an Encrypt-then-MAC-based construction or to a dedicated authenticated encryption construction. The former involves re-engineering the SSH BPP so that it first encrypts a message with some underlying encryption scheme and then MACs the resulting ciphertext. The latter involves modifying SSH to use a dedicated authenticated encryption scheme (e.g., XCBC [13], OCB [24]).

CONTINUE TO USE SSH-IPC? As mentioned, SSH-IPC is susceptible to an adaptive chosen-plaintext attack requiring an SSH user to encrypt on the order of 2^{13} packets. However, the attack may not be considered practical since it requires the attacker to, after seeing a ciphertext collision, control the *next* message that a user encrypts. If the session is encrypting a lot of data very quickly (e.g., while transferring a file), then an attacker may not have time to both recognize that a collision has occurred and to force the user to encrypt a specially-doctored message. Additionally, if we consider how the SSH transport protocol is used within SSH (and not as an entity by itself), then the attack is complicated by the fact that an application may compress and further encode user data before passing the resulting compressed payload to the SSH-IPC protocol. Nonetheless, we suggest that the use of SSH-IPC be deprecated. One simple reason is that, even if these attacks may be difficult to mount in practice, in the modern era of strong cryptography it would seem counterintuitive to voluntarily use a protocol with low security when it is possible to fix the security of SSH at low cost.

SWITCH TO SSH-NPC? Since SSH-NPC requires similar changes to the specification and implementations as SSH-$NPC while achieving less security than our other fixes, there does not appear to be any substantial reasons to switch to SSH-NPC. Therefore, we do not consider it further.

SWITCH TO SSH-$NPC? The advantages offered by SSH-$NPC are clear: it is provably secure and requires relatively minor and mostly localized changes to the SSH specification and to implementations. The added security, however, comes with the additional cost of up to two extra blocks per packet. In interactive sessions where an individual packet may only contain a few bytes of user data, the additional cost associated with those extra blocks may be significant

(in terms of bandwidth consumption, the time necessary to encrypt and MAC those two extra blocks, and the time required to generate the extra block of randomness). Another potential problem with SSH-$NPC is that it is prone to accidental implementation mistakes. Recall that if the padding used with SSH-$NPC is not randomized, then the same reaction attack against SSH-NPC will be effective here. Since two SSH implementations will inter-operate regardless of whether their padding is random or fixed, an SSH developer might accidentally use non-random or predictable padding. Such an accidental implementation mistake could have serious security consequences.

SWITCH TO SSH-CTR? SSH-CTRIV-CBC? OR SSH-EIV-CBC? The SSH-CTR instantiation is a promising candidate since it is provably secure, does not incur packet expansion, and does not require the padding to be random. Furthermore, there are several performance advantages with using CTR mode instead of CBC mode; for example, a software CTR mode implementation can be up to four times faster than a well-optimized CBC implementation [22]. Although perhaps not as attractive as SSH-CTR, SSH-CTRIV-CBC and SSH-EIV-CBC are also promising candidates because they also require no additional padding and because they only use one more block cipher invocation per packet than SSH-IPC.

Recall that the underlying encryption schemes for SSH-CTR, SSH-CTRIV-CBC, and SSH-EIV-CBC require both the sender and the receiver to maintain state. Prior to this work, most provable security analyses focused on encryption schemes with stateless decryption algorithms (hence our need to define security notions for encryption schemes with stateful decryption algorithms). Consequently, one initial objection to these three constructions might be that they require the underlying decryption algorithms to maintain state. However, since the composite SSH BPP decryption algorithm is already stateful (because the decoding algorithm is stateful), the fact that these three fixes use underlying encryption schemes with stateful decryption algorithms should be of little concern. Another potential disadvantage with CTR mode is that it is often perceived as being too "risky" [22]. As [22] points out, however, when used correctly and with proofs of security, CTR mode has many advantages over other encryption modes. Furthermore, as Bellovin and Blaze point out in [8], one can minimize the risk incurred with using CTR mode (including the risk of being forced to use repeating counters) if key management is done dynamically and properly, if it is not used with multiple senders who share keys, and if it is used in conjunction with strong integrity checks. All of these conditions hold in the case of SSH-CTR.

SWITCH TO ENCRYPT-THEN-MAC? Instead of insisting on using the current SSH Encode-then-E&M construction, it would also be possible to switch to another paradigm such as Encrypt-then-MAC (in which the message is first encrypted with an underlying encryption scheme and then the resulting ciphertext is MACed with an underlying message authentication scheme). This alternative is attractive because an Encrypt-then-MAC construction is provably secure assuming that its underlying encryption and message authentication schemes are also secure [5, 20]. We note, however, that since our recommended fixes provably meet our strongest notions of security, there may be little motivation to switch to an Encrypt-then-MAC-based construction. Additionally, switching to an Encrypt-then-MAC construction will likely require more intrusive modifications to the current SSH specification and to SSH implementations. Furthermore, unless care is taken, implementations of the modified SSH specification may not be compatible with implementations of the current SSH specification. Conceptually speaking, the changes incurred by SSH-CTR, SSH-$NPC, SSH-CTRIV-CBC, and SSH-EIV-CBC involve only changing the underlying encryption module and, in the case of SSH-$NPC, adding more random number generation for the padding. In contrast, the changes incurred by switching to the Encrypt-then-MAC construction involve changing the whole construction. Of course, the difference in the actual efforts that developers need to put in is highly implementation dependent.

SWITCH TO DEDICATED AUTHENTICATED ENCRYPTION SCHEMES? There are symmetric key-based authenticated encryption schemes that are designed from scratch and, thus, are potentially more efficient than schemes based on a black-box composition of off-the-shelf encryption and MAC components. These include RPC [19], XCBC [13], IACBC [18], and OCB [24]. Recall that currently the input to the SSH BPP's underlying encryption scheme is different from the input to the underlying MAC. There are two possible ways to incorporate a dedicated authenticated encryption scheme into SSH: (1) specifically re-design the SSH specification around a single authenticated encryption component or (2) somehow plug a dedicated authenticated encryption scheme into the current SSH design. As we mentioned when we considered the Encrypt-the-MAC paradigm, re-designing the SSH specification is probably not an attractive option.

For (2), the most logical way to incorporate a dedicated scheme into SSH would be to replace the current encryption scheme (CBC mode with chained IVs) with something like XCBC or OCB and to use the "none" message authentication scheme. As we argued for SSH-CTR, SSH-$NPC, SSH-CTRIV-CBC, and SSH-EIV-CBC, this modification should be fairly easy to do, and, given the efficiency of dedicated authenticated encryption schemes, could result in significant performance gains. The present drawback with this approach is that the current SSH specification does not include the 32-bit counter in the input to the underlying encryption scheme. Since, under this construction, the counter will not be bound to the input to the dedicated authenticated encryption scheme, this construction cannot protect against replay and out-of-order delivery attacks (while our proposed recommendations can). To rectify this situation, one would still have to modify more than just the "black-box" encryption component of the SSH BPP; doing so has the same drawbacks as possibility (1) above.

CLOSING REMARKS. We acknowledge that there are many possible ways to fix the current problems with the SSH protocol. We are biased towards our recommended fixes (e.g., SSH-CTR) because they are "less intrusive" than the other possible modifications but are still efficient and secure. "Less intrusive" is, however, a subjective measure and the IETF SSH working group may decide that it is feasible to re-engineer the SSH protocol to use an Encrypt-then-MAC-based construction or a dedicated authenticated encryption scheme. Given the inertia of the current SSH protocol, however, we feel that the working group may have a hard time

justifying significant modifications to the SSH specification. The goal of this work is to provide enough information to the SSH community so that the SSH community can make an informed decision when deciding how to fix the current problems with SSH.

Acknowledgments

The first and third authors were supported in part by NSF Grant CCR-0098123, NSF Grant ANR-0129617 and an IBM Faculty Partnership Development Award. The second author was supported by a National Defense Science and Engineering Graduate Fellowship. The second author also thanks the USENIX Association for a Student Grant supporting his earlier work with SSH. We thank Alexandra Boldyreva, Gregory Neven, Adriana Palacio, Bill Sommerfeld, and David Wagner for commenting on an earlier version of this paper.

8. REFERENCES

[1] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In L. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 83–107. Springer-Verlag, Apr. 1995.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proc. of the 38th FOCS*, pages 394–403. IEEE Computer Society Press, 1997.

[3] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. In Y. Desmedt, editor, *CRYPTO '94*, volume 839 of *LNCS*, pages 341–358. Springer-Verlag, Aug. 1994.

[4] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in SSH: Provably fixing the SSH Binary Packet Protocol. Cryptology ePrint Archive, Report 2002/078, 2002. `http://eprint.iacr.org/`.

[5] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer-Verlag, Dec. 2000.

[6] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 317–330. Springer-Verlag, Dec. 2000.

[7] S. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the 6th USENIX Security Symposium*, San Jose, California, July 1996.

[8] S. Bellovin and M. Blaze. Cryptographic modes of operation for the internet. In *Second NIST Workshop on Modes of Operation*, 2001.

[9] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 451–472. Springer-Verlag, 2001.

[10] W. Dai. An attack against SSH2 protocol, Feb. 2002. Email to the `ietf-ssh@netbsd.org` email list.

[11] DES modes of operation. National Institute of Standards and Technology, NIST FIPS PUB 81, U.S. Department of Commerce, Dec. 1980.

[12] W. Diffie and M. E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, Mar. 1979.

[13] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *FSE 2001*, LNCS. Springer-Verlag, 2001.

[14] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In R. Blakely, editor, *CRYPTO '84*, volume 196 of *LNCS*, pages 276–288. Springer-Verlag, 1985.

[15] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.

[16] C. Hall, I. Goldberg, and B. Schneier. Reaction attacks against several public-key cryptosystems. In *Proceedings of Information and Communication Security, ICICS'99*, 1999.

[17] Internet Engineering Task Force. Secure Shell (secsh) charter, 2002. `http://www.ietf.org/html.charters/secsh-charter.html`.

[18] C. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 529–544. Springer-Verlag, May 2001.

[19] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 284–299. Springer-Verlag, Apr. 2000.

[20] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 310–331. Springer-Verlag, Aug. 2001.

[21] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authenticationa. IETF Internet Request for Comments 2104, Feb. 1997.

[22] H. Lipmaa, P. Rogaway, and D. Wagner. CTR-mode encryption. In *First NIST Workshop on Modes of Operation*, 2000.

[23] P. Rogaway. Problems with proposed IP cryptography, 1995. Available at `http://www.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt`.

[24] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *Proc. of the 8th CCS*, pages 196–205. ACM Press, 2001.

[25] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Tenth USENIX Security Symposium*, 2001.

[26] S. Vaudenay. Security flaws induced by CBC padding – applications to SSL, IPSEC, WTLS .... In L. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–545. Springer-Verlag, 2002.

[27] T. Ylonen. SSH — Secure login connections over the Internet. In *Sixth USENIX Security Symposium*, 1996.

[28] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH transport layer protocol, 2002. Draft 12, available at [17].