Formally Modeling the Terrapin Attack on SSH

Jörg Schwenk[1 [0000−0001−9315−7354]], Fabian Bäumer[1 [0009−0006−5569−6625]], and Marcus Brinkmann[1 [0000−0001−5649−6357]]

Ruhr University Bochum, Bochum, Germany

Abstract. The Terrapin attack against SSH channel integrity (USENIX Security 2024) used a novel attack vector: attacks on the *channel state*. Surprisingly, not all AEAD modes of SSH were equally affected by this attack, and it remained an open question if "unaffected" meant "secure". Existing formal models for secure channels are based on stateful encryption. However, these models do not define what the channel state is and how it is used as input to the different AEAD modes.
In this paper, we propose a formal model for channel integrity under partially chosen state. Applied to the Terrapin attack, the chosen state is the SSH sequence number. It uses an abstract stateful encryption interface, for which we provide pseudocode descriptions for the eight most prominent AEAD modes used in SSH. By varying the SND oracle, we can model ciphertext-only (CO; the Terrapin attack), known-plaintext (KPA), and chosen-plaintext (CPA) attacks. This allows us to establish concrete bounds on the security of the AEAD modes. We find that all three Encrypt-then-MAC (EtM) modes and ChaCha20-Poly1305 in SSH are insecure in the CO model. AES-GCM is the only cipher secure in all three model variants. Going beyond Terrapin, we show that Encrypt-and-MAC (EaM) with a CBC cipher is secure, even in the KPA model. In particular, we describe a novel BEAST-like chosen-plaintext attack on the channel integrity of EaM-CBC, which separates the KPA and CPA models for this scheme.

Keywords: AEAD · Stateful Encryption · Formal Analysis · Adversary-chosen State · Terrapin Attack

1 Introduction

1.1 Motivation

Terrapin Attack. In 2024, an attack on the SSH protocol was published [5], which attracted interest from both academia and industry. One insight from that paper is that the stateful encryption schemes deployed in SSH are affected differently by the attack. While the channel integrity for ChaCha20-Poly1305 and EtM is broken, GCM and EaM modes are unaffected. This is somehow counterintuitive: if the main channel state can be controlled by an adversary, why is there this difference? And what does "unaffected" mean: does it mean "secure" in any formal sense? While the informal treatment in [5] has the advantage that

it can also include the application layer, a rigorous answer to these questions requires a formal analysis of the underlying constructions. Thus we raise the question:

RQ 1: Can we extend existing security models for secure channels to include Terrapin-like attacks on channel state and use these models to explain the differences in the security of SSH cipher modes?

SSH. SSH 2.0, similar to TLS, establishes an authenticated confidential channel between a client and a server. The secure channel uses the Binary Packet Protocol (BPP) and stateful encryption schemes for the packets. The BPP, key exchange, and server authentication are described in [32], while two other documents specify client authentication [29] and the application-layer protocol [30]. Cipher modes in SSH may differ significantly from other protocols; a striking example is ChaCha20-Poly1305, where the SSH version (Fig. 10) structurally differs from RFC 8439 [39]; thus we describe each mode in detail in Section 4.

The Terrapin attack [5] is enabled by two design flaws in SSH: (1) The SSH handshake allows for many optional messages, which are not included in the hash of the partial handshake transcript; (2) SSH sequence numbers are not reset when key material changes. In the attack, the attacker manipulates the sequence numbers by injecting optional messages into the handshake and resynchronizes the sequence numbers by removing an equal number of messages at the beginning of the secure channel (see Section 2.5). In this paper, we model the handshake message injection vector as an oracle available to the attacker and formally prove the consequences for eight different SSH stateful encryption schemes.

Secure Channels, AEAD and Stateful Encryption. Secure channels, like the TLS Record Layer and the SSH BPP, are constructed from *AEAD cipher modes* by using channel-specific *state information*—typically sequence numbers. In fact, Rogaway [47] identified secure network channels as the origin of AEAD. Theoretical constructions of secure channels have been described, among others, by Boyd et al. [9] and Fischlin et al. [16]. However, the cryptographic primitive used in these constructions is not AEAD but *stateful encryption*, a notion introduced by Bellare, Kohno, and Namprempre [6] and refined in [20,27,9,16].

Unfortunately, the interfaces of these two primitives are different; see Table 1. AEAD input consists of a secret key k, a unique nonce N, associated data A, and plaintext m; the output is a ciphertext c and an authentication tag τ. In stateful encryption, a key k, a channel state st, and a message m are the inputs; the output is an authenticated header H, the ciphertext c, and an updated state st'. These differences make it challenging to compare formal models for secure channels with their real-world implementations. Thus we ask:

RQ 2: Can we map stateful encryption schemes to AEAD cipher modes to include these modes in the analysis of secure channels?

Table 1. Syntax for AEAD [47,34] and Stateful Encryption [20,27].

Paradigm	Encryption Syntax
AEAD	$(c, \tau) \leftarrow \mathsf{AEAD.Enc}(\mathsf{k}, N, A, m)$
Stateful Enc.	$(\mathrm{H}, c, \mathsf{st}') \leftarrow \mathsf{SF.Enc}(\mathsf{k}, \mathsf{st}, m)$

We answer RQ 2 in the affirmative: in Table 2, we classify the input data for eight different SSH stateful encryption schemes. We provide pseudocode for the encryption in Figs. 4, 6, 7 and 9. There, we show which values from Table 2 are provided via the stateful encryption interface and how the AEAD input data (especially A and N) are constructed. Likewise, we show how the AEAD function is called and how the state is updated. Thereby, we introduce a pseudocode translation between the stateful encryption and AEAD cipher modes.

Formal Models and the Terrapin Attack. When we first analyzed the differences in resilience against the Terrapin attack among the SSH stateful encryption schemes, we found that ChaCha20-Poly1305 is vulnerable because it has no state except for the adversary-chosen sequence numbers. But we also found that the presence of additional state does not imply security against the attack: EtM schemes are vulnerable, while EaM schemes are not. Thus, we ask:

> *RQ 3: Can we prove that SSH stateful encryption modes* "unaffected" *by Terrapin are secure in our model? Can we further generalize our security model to characterize the minimum attacker capabilities required for a successful attack?*

We answer it affirmatively and go beyond modeling the Terrapin attack by introducing three different SND oracles:

1. The Terrapin attack is the strongest attack because it only assumes a weak CO adversary, which we model through a $\mathsf{SND\text{-}CO}$ oracle. In the corresponding experiment, we can formally reproduce the results from the Terrapin paper with a notable difference: In our model, EtM schemes are unconditionally vulnerable, while in [5], these schemes are classified as *"affected with limited exploitability"* because corrupted plaintext leads to failures at the application layer with high probability. Another difference is that we also apply the analysis of this model to EtM-Stream and EaM-Stream schemes.
2. Standard models for secure channels [6,20,27,9,16,41,48,25] require security under CPA from the channel, which we model through a $\mathsf{SND\text{-}CPA}$ oracle, where the adversary has full control over the plaintext. We show that in this scenario, all EaM modes become insecure. This is straightforward for EaM-CTR and EaM-Stream, since both are essentially stream ciphers and encryption is not protected by the MAC (Section 5.4). For EaM-CBC, we present a new attack using BEAST-like techniques [46] to break channel integrity (Section 5.6).

3. In between CO and CPA, we consider security under KPA, which we model with a SND-KPA oracle, where the adversary learns the plaintext but cannot choose it. We leverage this model to differentiate between the EaM modes: in this model, our new attack does not work against EaM-CBC, and we can prove the security of this mode, but EaM-CTR and EaM-Stream remain insecure.

1.2 Contributions

- We present the first formal security model for attacks under partially chosen channel state, which is syntax-compatible with previous work on secure TCP channels (Section 3). We model state manipulation by introducing SET oracles. In the CO variant, which captures the Terrapin attack, we prove that all EtM modes and ChaCha20-Poly1305 are insecure. Going further, we prove that the *"unaffected"* modes GCM, EaM-CBC and EaM-CTR are in fact CO-secure. We also cover two Stream-based schemes not analyzed in [5].
- We extend our model to KPA and CPA adversaries to find the minimum attacker capabilities required to break each scheme. We prove that GCM remains secure even against CPA, and EaM-CBC against KPA, while EaM-CTR and EaM-Stream are KPA-insecure. For EaM-CBC we give a novel CPA attack on channel integrity, based on techniques from the BEAST attack [46]. This yields a strict separation between the EaM modes that the informal analysis in [5] could not make (Section 5).
- We document eight stateful encryption schemes used in SSH in cryptographic pseudocode, making them accessible to formal analysis. We show that they are vastly different and even deviate from the AEAD syntax (Section 4).
- We harmonize AEAD and stateful encryption interfaces by distinguishing derived input values, randomly chosen values, channel state, associated data, and plaintext (Table 2).

2 Background

2.1 Notation

In this paper, we use mostly standard notation. $a \| b$ denotes concatenation of a and b, ε is the empty word; $a, b \leftarrow c$ is a shorthand for $a \leftarrow c$ and $b \leftarrow c$. $a \leftarrow\!\!\$\ \mathcal{S}$ denotes randomly sampling an element from set $\mathcal{S}$. $\mathsf{Pad}_{\mathsf{SSH}}(\cdot)$ and $\mathsf{Pad}^{*}_{\mathsf{SSH}}(\cdot)$ are the SSH padding functions described in Section 2.2. $\mathsf{len}(a)$ denotes the byte length of a. $\mathsf{LastBlock}(c)$ returns the last cipher block of c. $\texttt{0xff}^i$ is the octet string that consists of i repetitions of the byte `0xff`. $s_{i\ldots j}$ are the $j - i + 1$ bytes from index i to j (inclusive). n_{Enc} denotes the block length of a cipher in bits.

2.2 SSH

The core features of SSH 2.0 are specified in RFCs 4250–4254 [28,31,29,32,30]. Several documents describe EaM modes. RFC 4253 [32] specifies RC4- and CBC-based modes, RFC 4344 [38] specifies CTR-based modes, and RFC 4345 [17]

specifies additional RC4-based modes. While RC4 has been deprecated in SSH in RFC 8758 [50], it may still be in use. The EtM modes are described—very briefly—in [40]. ChaCha20-Poly1305 was added by an OpenSSH specification and was recently adopted by the IETF sshm working group [37], while GCM is specified in RFC 5647 [18] and [35] (AES only).

SSH Key Exchange. RFC 4253 describes the SSH key exchange up to the establishment of the secure channel. This key exchange is initiated with an ASCII-over-TCP banner exchange with no prescribed order. This is followed by two KexInit messages contained in unprotected *Binary Packet Protocol* (BPP) packets; here, BPP only adds some headers. These messages contain lists of cryptographic algorithms. Both client and server use a deterministic algorithm to derive the negotiated algorithms from these lists. Again, there is no predefined order; the client and server may proceed as they wish. The next two messages implement a Diffie-Hellman Key Exchange (DHKE), and the server adds a signature over a partial transcript of the handshake to authenticate itself. By sending Newkeys messages, both parties activate the secure BPP channels using the negotiated algorithms and keys. Client authentication (RFC 4252, [29]) is done later and is irrelevant for the Terrapin attack.

SSH Secure Channel. RFC 4253 specifies BPP as a secure EaM channel (cf. [8]): each plaintext packet consists of a length header, a padding length header, the plaintext, and random padding. The length field counts the padding length field, the payload, and the padding, but neither itself nor the MAC. The ciphertext is computed over both length fields, the payload and the padding. The MAC is computed over the sequence number followed by the same fields, and is appended in the clear. Please note that to mitigate the attack described in [2], this structure has been significantly altered in different ways; for details see section 4.

SSH Padding. SSH padding plays an important role in determining the exact probabilities in some attacks and proofs. In RFC 4253 [32], the padding of plaintext messages m is described for EaM modes: m is padded by appending $n \geq 4$ random bytes and prepending n as a 1-byte padding length field, where n is chosen such that $\mathsf{len}(m) + n + 5$ is a multiple of $B = \max(n_{\mathsf{Enc}}/8, 8)$. For stream ciphers, this yields an 8-byte alignment. We denote this process as $\mathsf{Pad}_{\mathsf{SSH}}(m)$, whose output has length $\mathsf{len}(m) + n + 1$. The remaining 4 bytes account for the packet length field, which $\mathsf{Pad}_{\mathsf{SSH}}(m)$ does not emit but which is prepended separately and, under EaM, counted toward the alignment because it is encrypted alongside m. For non-EaM cipher modes, the padding differs slightly: $\mathsf{len}(m)+n+1$ is aligned instead of $\mathsf{len}(m)+n+5$. The 4-byte difference is the packet length field, which here is sent unencrypted or encrypted under a separate cipher instance and thus excluded from the alignment. We denote this variant as $\mathsf{Pad}^{*}_{\mathsf{SSH}}(m)$.

The padding length of SSH is inherently limited by the 1-byte padding length field and B. This clashes with asymptotic proofs because, while tiny, the probability of an adversary $\mathcal{A}$ guessing the padding is non-negligible in λ. We therefore make the reasonable assumption that $B = \Omega(\lambda)$, corresponding to a growing n_{Enc} and minimal alignment. Consequently, $\mathsf{len}(\mathsf{Pad}_{\mathsf{SSH}}(m)) = \Omega(\lambda)$.

2.3 AEAD Ciphers

Bellare and Namprempre [7,8] list three generic AEAD construction modes. For block ciphers, there is an additional padding step, which is omitted below.

- **MAC-then-Encrypt (MtE).** In this mode, a MAC τ is computed over the plaintext, concatenated with it, and the combination is encrypted. This is the default AEAD mode in TLS 1.2.
- **Encrypt-and-MAC (EaM).** The MAC τ is computed over the plaintext, but only the plaintext is encrypted. τ is then attached to the ciphertext. This is the default mode for SSH in [32].
- **Encrypt-then-MAC (EtM).** First, the plaintext is encrypted, then the MAC τ is computed over the ciphertext. For SSH, this mode is available, and it is the default mode in TLS 1.3.

Motivated by existing constructions in secure network channels, Rogaway [47] introduced the notion of *AEAD*. He summarized the known generic constructions and added two novel ones: *nonce stealing* and *ciphertext translation*. He proposed an interface in which the input for AEAD encryption is specified as a 4-tuple of a single encryption key k, a nonce N, associated data A, and plaintext m. The IETF adopts this interface in [34], and nonce stealing is widely used in specifications.

Definition 1 (AEAD). *An AEAD (Authenticated Encryption with Associated Data) scheme is a triple* $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $\mathsf{k} \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$ *generates a key* k *given a security parameter* λ*;*
- $c\|\tau \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{k}, N, A, m)$ *takes a secret key* k*, a nonce* N*, associated data* A*, and a plaintext* m*, and outputs a ciphertext* c *and authentication tag* τ*;*
- $m \leftarrow \mathsf{Dec}(\mathsf{k}, N, A, c\|\tau)$ *takes a secret key* k*, a nonce* N*, associated data* A*, and a ciphertext* $c\|\tau$*, and returns either the plaintext* m *or a failure symbol* $\perp$*.*

2.4 Stateful Encryption Schemes

While Rogaway [47] analyzed the construction of single network packets to derive the abstraction of AEAD, Bellare, Kohno, and Namprempre [6] analyzed—using SSH as an example—how the *sequence* of network packets can be protected from replay, reordering, and packet deletion attacks. In practice, this is achieved by using sequence numbers in AEAD. The authors formalized such constructions as *stateful encryption*. In these constructions, encryptions of subsequent packets depend not only on the key, randomization, and plaintext, but also on some *state* inherited from the previous packet encryption.

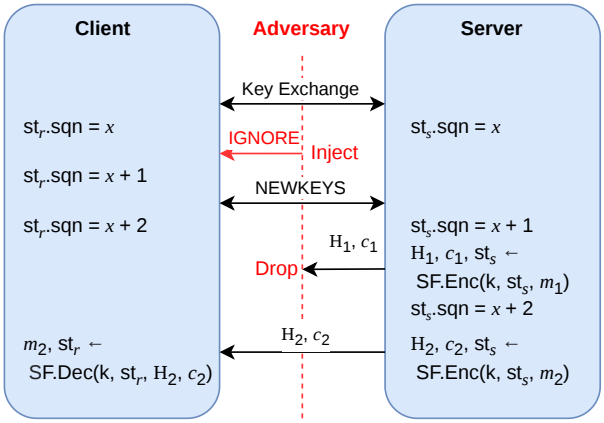

Fig. 1. Illustration of the Terrapin attack on the BPP from [5]. The server sends m_1 and m_2, but the client receives only m_2.

Definition 2 (Stateful Encryption Scheme). *A* stateful encryption scheme *is a triple* $\mathsf{SF} = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Dec})$ *where:*

- $(\mathsf{k}, \mathsf{st}_0) \leftarrow\!\!\$\ \mathsf{Init}(1^\lambda)$ *generates a key* k *and an initial state* st_0 *given a security parameter* λ*;*
- $(\mathrm{H}, c, \mathsf{st}_{i+1}) \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{k}, \mathsf{st}_i, m)$ *takes a secret key* k*, the state* st_i*, and a message* m*, and outputs a header* H*, a ciphertext* c*, and an updated state* st_{i+1}*;*
- $(m, \mathsf{st}_{j+1}) \leftarrow \mathsf{Dec}(\mathsf{k}, \mathsf{st}_j, \mathrm{H}, c)$ *takes a secret key* k*, the state* st_j*, a header* H*, and a ciphertext* c*, and outputs a message* m *(or* $\perp$ *in case of failure), and an updated state* st_{j+1}*.*

W.l.o.g., each state is partitioned as $\mathsf{st} = (\xi, \eta)$ *where* $\xi \in \mathcal{S}_{\mathsf{adv}}$ *is the adversary-chosen state and* $\eta \in \mathcal{S}_{\mathsf{sec}}$ *is the secure state.*

The abstraction of stateful encryption has been used to analyze the security of protocols [20,27,16] and to develop theoretical constructions [9].

2.5 Terrapin Attack

The Terrapin attack [5] is illustrated in Fig. 1. It exploits two design flaws of the SSH Transport Layer Protocol: first, the server's signature in the second of the two key exchange messages does not cover the whole handshake transcript, but only a subset thereof, allowing injection of optional messages (e.g., IGNORE) that may be sent at any time during the key exchange; and third, in contrast to TLS, SSH sequence numbers are not reset when the key exchange concludes.

This enables the *prefix truncation attack* on the secure channel of SSH depicted in Fig. 1 (with $x = 2$): The Man-in-the-Middle (MitM) adversary injects an optional and ignored message, such as IGNORE, into the unprotected BPP—since no encryption or message integrity is yet active, the resulting change in

the client's receiving sequence number from 2 to 3 remains undetected. After receiving the legitimate NEWKEYS message from the server, the value is incremented to 4, which is then used to validate the integrity of the first encrypted message received by the client. When the server sends the NEWKEYS message, it increments the corresponding sending sequence number to 3 and, after sending (H_1, c_1), to 4. The MitM adversary then drops (H_1, c_1) during transmission; this is the aforementioned *prefix truncation*. During the stateful encryption of (H_2, c_2), the server uses sequence number 4 as part of the channel state for the server-to-client unidirectional channel. When receiving (H_2, c_2), the client uses the same value to perform the stateful decryption, which consequently succeeds, leaving the adversary's prefix truncation undetected by the recipient.

3 Model

We define a model for *channel integrity under partially chosen state attacks* (CINT-PCSA). Channel integrity addresses both forged AEAD ciphertexts and reordering of messages within the channel. We formally define the security of a stateful encryption scheme as a security game played between a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ that models a single unidirectional BPP channel described as a stateful encryption scheme. The capabilities of the adversary are specified in Fig. 2 as pseudocode oracles. The SND oracle models the sender side of the channel and the RCV oracle models the receiver; the sender and receiver have already agreed on a common key k and initial channel state $\mathsf{st}_s = \mathsf{st}_r$. The adversary may query these oracles with any inputs and receive the output values; this models that the adversary has a MitM position. In addition to these *network privileges*, $\mathcal{A}$ gets one *plaintext privilege* (ciphertext-only/CO, known plaintext/KPA, chosen plaintext/CPA) through the different $\mathsf{SND\text{-}X}$ instantiations, where $\mathrm{X} \in \{\mathrm{CO}, \mathrm{KPA}, \mathrm{CPA}\}$. Finally, $\mathcal{A}$ gets *partially chosen state privilege* through the SET oracles.

The goal of $\mathcal{A}$ is to compute the challenge bit b; if its result b' equals b, $\mathcal{A}$ *wins* the game. Since a randomly chosen bit b can be guessed with probability $1/2$, winning the security game does not immediately render a stateful encryption scheme insecure; therefore, we eliminate the effect of guessing by measuring the absolute difference between the winning probability of $\mathcal{A}$ and the guessing probability $1/2$ and call this the *advantage* of $\mathcal{A}$. Since b is only used in RCV, the adversary can only increase its advantage by calling RCV.

3.1 Experiment

Exp. During initialization, $\mathcal{C}$ calls $\mathsf{SF.Init}$ to generate a symmetric key k uniformly at random alongside an initial state st_s.[1] By definition, st_s consists of an adversary-chosen state ξ and secure state η. $\mathcal{C}$ then duplicates the initial state

[1] In real-world protocols, an AKE (Authenticated Key Exchange) protocol is used for agreement on a key k and possibly an initial state st_s, for example, the key exchange in the SSH Transport Layer Protocol.

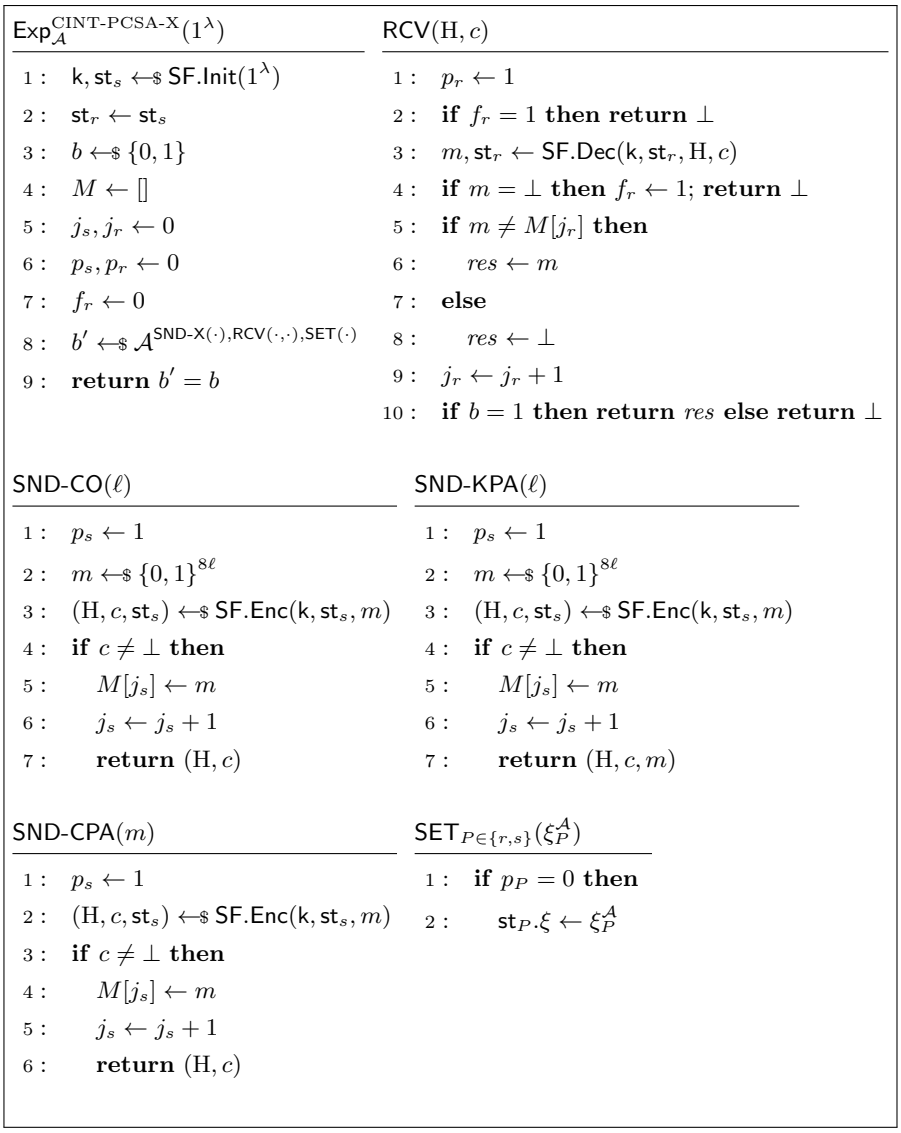

Fig. 2. Our model for channel integrity under partially chosen state attacks. The pseudocode defines a single-instance security game played between an adversary $\mathcal{A}$ and a CINT-PCSA challenger, where the adversary has access to both SET oracles, the RCV oracle, and one of the SND oracles. The different SND oracles model the adversary's knowledge of and control over the plaintext: ciphertext-only (CO), known plaintext (KPA), or chosen plaintext (CPA).

for the RCV oracle as st_r, thereby modeling synchronized initial encryption and decryption states. The structure of the state depends on the stateful encryption scheme used; in all instantiations considered in this paper, the sequence number sqn forms the adversary-chosen state and initialized to zero (cf. Table 2).

Besides that, we need bookkeeping variables. First, there is the challenge bit b. This bit is an artifact used to compute the advantage of the adversary in the security game and does not occur in real-world instantiations. $\mathcal{C}$ also initializes the sent and received counters, j_s and j_r (not to be confused with sequence numbers), which point to entries in an array M, to zero. The array M is used by $\mathcal{C}$ to track the adversary's queries to SND and to check the correct order of decryption in RCV. Every entry in M is initialized just-in-time; accessing an uninitialized array index returns $\perp$. We do not allow adaptive calls to SET in our model, which is implemented by the flags p_s and p_r. The adversary can set a corresponding adversary-chosen state only before the first call to SND or RCV has occurred; when SND or RCV is called, we block the adversary from using the corresponding SET oracle by setting p_s or p_r. After the initialization, the adversary $\mathcal{A}$ is provided access to one of the three SND oracles, as well as RCV and both SET oracles. After polynomially many steps, the adversary returns a bit b'. The adversary wins the game if $b' = b$.

Definition 3. *For* $\mathrm{X} \in \{\mathrm{CO}, \mathrm{KPA}, \mathrm{CPA}\}$*, a stateful encryption scheme* SF *is* CINT-PCSA-X*-secure if, for every PPT adversary* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}_{\mathcal{A}}^{\text{CINT-PCSA-X}}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

is negligible in λ*.*

SND. SND comes in three different flavors. SND-CO allows the adversary to choose the byte length ℓ of the message and outputs the authenticated header H and ciphertext c of a randomly chosen message m. SND-KPA returns the randomly chosen message m in addition to (H, c). Finally, SND-CPA allows $\mathcal{A}$ to choose m as in a CPA oracle. In each case, the message m is encrypted by calling the stateful encryption scheme; afterward, the state variable st_s is updated accordingly. The stateful encryption scheme can be instantiated with each of the eight schemes from Table 2 by using the appropriate pseudocode interface from one of the Figs. 4, 6, 7 and 9. If the stateful encryption succeeds, we store the plaintext m in the next free slot of the *sent array* M, indexed by j_s. Afterward, we increase the sent counter j_s, and finally return (H, c) to the adversary $\mathcal{A}$.

RCV. RCV is a decryption oracle that accepts chosen header data H and any ciphertext c as input. c can be a ciphertext previously issued by SND—in which case AEAD decryption will most likely succeed—or a ciphertext generated or modified by $\mathcal{A}$—in which case stateful decryption will most likely return an error symbol $\perp$ caused by a MAC/tag verification failure. Because of the **else** condition on line 7, in most cases, the value $\perp$ will be returned. The only exception is when

c decrypts to a novel plaintext or when decryption happens out of order. This is checked by the **if**-clause in line 5. The RCV oracle checks whether the plaintext returned on the (j_r+1)-th successful decryption call is *not* equal to the plaintext entry in M at index j_r.

By definition of the security game, MAC forgeries or out-of-order deliveries are not *directly* considered winning events. However, when such an event happens and $b = 1$, $\mathcal{A}$ gets precise information about b and can thus maximize its advantage. Whenever a plaintext $m \neq \perp$ is returned, $\mathcal{A}$ *knows* that $b = 1$ and can set $b' = 1$. This is why the last, seemingly strange, check is implemented in line 10 of RCV. The benefit of this advantage-based security notion is that our game can easily be extended to a full game for channel security, also covering IND-CCA2 security (cf. Fig. 11 in the Appendix), without changing the syntax of the model. This follows a line of research in channel security initiated by [41,20,27].

3.2 Restrictions

Application Layer. We do not model the application-layer protocols to which the plaintexts of the secure channel are delivered. Any novel or out-of-order plaintext m' accepted by RCV constitutes a successful attack, regardless of whether it is a valid application-layer message. This is why some "unsuccessful" attacks on SSH in the Terrapin paper are successful in our model—the Terrapin authors *did* take the application layer of SSH into account.

IND-CCA2. Comparing Fig. 2 and Fig. 11, it is clear that we are *not* considering IND-CCA2 security in this paper. The reasons are twofold:

1. Our goal is to determine unambiguously whether an SSH cipher mode is *secure* against the Terrapin attack. Including IND-CCA2 security in our security model would make this determination no longer possible. For example, EaM-CBC is secure against the Terrapin attack but not IND-CCA2-secure, as shown in [2]. Consequently, we would have to label it as "insecure".
2. The IND-CCA2 status of some cipher modes is not yet known. While some results have been presented in [42] and [1], not all modes are covered yet. This would have required the category "unknown" in the classification.

3.3 Differences to Formal Models in Related Work

Bellare, Kohno, and Namprempre [6] model the indistinguishability of ciphertexts and the integrity of plaintexts/ciphertexts in two separate security games. We use an idea from Paterson, Ristenpart, and Shrimpton [41], who combine the two parts into one advantage-based experiment (Fig. 11), except that we omit the IND-CCA2 component in Fig. 2 to focus on the Terrapin attack.

We do not follow [41] in allowing the adversary to control the amount of padding, since many libraries do not expose this functionality to the application

layer. Thus, the ℓ parameter has a different meaning in our model—it specifies the length of a plaintext, not of the ciphertext.

Several models use different levels of checks on the ordering of messages. Rogaway and Zhang [48], Boyd et al. [9], and Kohno, Palacio, and Black [25] define four levels: L_1 (no message injections), L_2 (no replays), L_3 (no reordering), and L_4 (receiving order must match sending order). A fifth level, L_5, defined in the appendix of [25], differs from L_4 only in the ability of the adversary to send decrypt queries after a failed attempt, so L_5 matches theoretical models [41], while L_4 better describes SSH in practice. As our goal is to determine whether a given stateful encryption scheme is vulnerable to Terrapin, we implement L_4 in our model. By removing the failure flag f_r, the model can be converted to an L_5 model. In [48], a new approach is proposed to unify security models for stateful encryption. This is motivated by the complex and often counterintuitive security definitions for the different channel security levels presented in the papers cited above. Rogaway and Zhang use correctness definitions of the different channels to define the notion of *indistinguishability up to correctness*. We hope that by focusing exclusively on TCP-based secure channels, the complexity of our model remains manageable.

Jager et al. [20] use a multi-instance stateful variant of the model from [41] to model ACCE. This is justified because in TLS, many stateful encryption channels are instantiated in parallel and an active adversary may exploit this.

3.4 Additional Related Work

AEAD. Katz and Yung [22] proposed one of the first AEAD modes. Generic constructions for AEAD were investigated by Krawczyk [26]. The IND-CCA3 definition from Shrimpton [49] is a predecessor of the unified security definition for AEAD given by Paterson, Ristenpart, and Shrimpton [41]. **ACCE.** In the analysis of all three cipher suite families from TLS 1.2, Krawczyk, Paterson, and Wee [27] and Jager et al. [20,21] use the ACCE model, which combines authenticated key agreement with stateful AEAD. **Data Streams.** Fischlin et al. [16] refined the stateful channel security model further to the data *streams* sent and received by the application layer, which would be partitioned into packets solely by the sender's buffer. They gave the adversary access to a *flush* interface, allowing the adversary to artificially partition the application data stream into packets. **Constructions.** The above works provide abstract formal security games to analyze the security of existing real-world implementations of protocols like SSH and TLS. Attacks on these protocols revealed that the abstract security model did not cover all subtleties of the protocol implementations. Subsequently, models were adapted to the protocol implementations (TLS: Paterson, Ristenpart, and Shrimpton [41]; SSH: Albrecht, Paterson, and Watson [2], as well as Paterson and Watson [42]), or implementations were adapted to models after attacks (SSH: Bäumer, Brinkmann, and Schwenk [5], Miller [36]). Delignat-Lavaud et. al [12] modeled different TLS 1.3 AEAD modes; their model does not include channel state.

Table 2. Supported stateful encryption schemes and their parameters in SSH. The columns CO, KPA, and CPA indicate if a given scheme is secure in the corresponding security model. In SSH, each connection consists of two unidirectional channels, each with a separate state and keying material. The adversary-chosen state component is ξ, and the secure state component is η. For GCM, sqn is maintained but not used.

SSH Cipher	[5] (informal)	H	$\mathsf{st}.\xi$	$\mathsf{st}.\eta$	Key Derivation	Fig.	CO	KPA	CPA	Proofs
EaM-CBC	unaffected	–	sqn	IV	k_e, k_a, $\mathsf{IV}_{\mathsf{init}}$	3, 4	✓	✓	✘	5.6, 5.7
EaM-CTR	unaffected	–	sqn	ctr	k_e, k_a, $\mathsf{ctr}_{\mathsf{init}}$	3, 4	✓	✘	✘	5.4, 5.5
EaM-Stream	–	–	sqn	pos	k_e, k_a	3, 4	✓ [1]	✘	✘	5.4, 5.5
EtM-CBC	affected	$\mathsf{len}(m')$	sqn	IV	k_e, k_a, $\mathsf{IV}_{\mathsf{init}}$	5, 6	✘	✘	✘	5.2
EtM-CTR	affected	$\mathsf{len}(m')$	sqn	ctr	k_e, k_a, $\mathsf{ctr}_{\mathsf{init}}$	5, 6	✘	✘	✘	5.2
EtM-Stream	–	$\mathsf{len}(m')$	sqn	pos	k_e, k_a	5, 6	✘	✘	✘	5.2
GCM	unaffected	$\mathsf{len}(m')$	sqn	ictr	k, salt, $\mathsf{ictr}_{\mathsf{init}}$	7, 8	✓	✓	✓	5.3
CCP [2]	affected	–	sqn	–	$\mathsf{k} = (\mathsf{k}_2, \mathsf{k}_1)$	9, 10	✘	✘	✘	5.1

[1] Unknown for RC4 [2] ChaCha20-Poly1305 as stateful encryption, SSH variant

4 Channel State and AEAD Cipher Modes

In this section, we analyze the stateful encryption schemes supported by SSH and how they construct the AEAD input N and A from the message m and the channel state st. Furthermore, we detail the structure of the authenticated header H. An overview is given in Table 2.

4.1 Preliminaries

Associated Data and Channel State. As Table 2 shows, for all described stateful encryption schemes in SSH, *channel state* always consists of the sequence numbers as an adversary-chosen state component and may have a non-empty secure state component depending on the underlying AEAD mode. Thus, channel state may influence MAC validation directly (when included in A) or indirectly (for INT-PTXT security, by changing the decryption). The inclusion method may differ; e.g., a sequence number may either be concatenated with the packet header (EaM, EtM), or used with nonce stealing (ChaCha20-Poly1305).

Historically, after the dissolution of the original secsh working group and before the formation of the sshm working group by the IETF, SSH development was primarily driven by OpenSSH. Of the eight commonly supported stateful encryption schemes in SSH, only half are described in RFCs; these include the three generic modes using EaM constructions described in [32,38]. The other half, including the three generic modes using EtM constructions, are described in separate documents [40,37].

4.2 Encrypt-and-MAC

The structure of this encryption mode is illustrated in Fig. 3, and the pseudocode translation from stateful encryption to AEAD in Fig. 4.

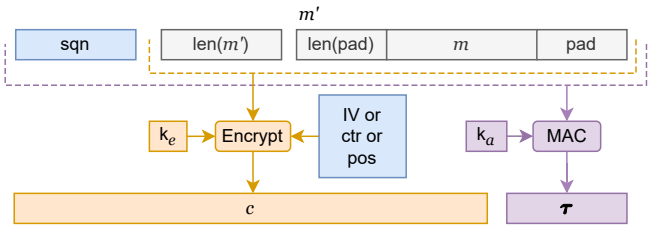

Fig. 3. Encryption of a binary packet in SSH using a generic Encrypt-and-MAC construction [32].

CBC. In SSH, when using the EaM construction with a CBC block cipher ([32, Sec. 6.3]), the encoded plaintext m' consists of the packet length, the padding length, the message m, and the random padding bytes. Consequently, the header H returned by the stateful encryption is empty.

SSH uses IV chaining: the initial value $\mathsf{IV}_{\mathsf{init}}$ is generated during key derivation, and after each encryption IV is set to the last block of the previous ciphertext. The channel state thus consists of the sequence number as adversary-chosen and the IV as secure state. The MAC τ is computed over the encoded plaintext m', prepended with the sequence number sqn.

If the wrong sequence number is used for decryption, MAC verification and thus the stateful decryption operation overall will fail. If the wrong IV is used for stateful decryption, the plaintext produced by the block cipher will change. Since the MAC is computed over the plaintext, MAC verification, and thus stateful decryption, will also fail.

CTR. In SSH, when using the EaM construction with a CTR block cipher ([38, Sec. 4]), the counter used for encryption has the size of the block length n_{Enc} (128 bits for AES). Since the counter is never reset under the same key material, it forms the secure state component. The initial value of the counter is generated through key derivation similar to the initial value of the IV in CBC mode, that is, $\mathsf{ctr}_{\mathsf{init}} = \mathsf{IV}_{\mathsf{init}}$. After each block, the corresponding counter is incremented.

Stream. SSH supports three algorithms based on the RC4 stream cipher: `arcfour` [32] is classical RC4 with a key size of 128 bits; `arcfour128` and `arcfour256` [17] are variants with key sizes of 128 and 256 bits; additionally, the first 1536 bytes of output from the Pseudorandom Generator (PRG) are

$\mathsf{SF.Enc}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}CBC}}((\mathsf{k}_e, \mathsf{k}_a), (\mathsf{sqn}, \mathsf{IV}), m)$

// $\mathsf{sqn} \in \{0,1\}^{32}, \mathsf{IV} \in \{0,1\}^n$

1 : $m' \leftarrow\!\$\ \mathsf{Pad}_{\mathsf{SSH}}(m)$
2 : $m' \leftarrow \mathsf{len}(m') \| m'$
3 : $c \leftarrow \mathsf{CBC.Enc}(\mathsf{k}_e, \mathsf{IV}, m')$
4 : $A \leftarrow \mathsf{sqn}$
5 : $\tau \leftarrow \mathsf{MAC}(\mathsf{k}_a, A \| m')$
6 : $\mathsf{sqn}' \leftarrow \mathsf{sqn} + 1$
7 : $\mathsf{IV}' \leftarrow \mathsf{LastBlock}(c)$
8 : **return** $(\varepsilon, c \| \tau, (\mathsf{sqn}', \mathsf{IV}'))$

$\mathsf{SF.Enc}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}CTR}}((\mathsf{k}_e, \mathsf{k}_a), (\mathsf{sqn}, \mathsf{ctr}), m)$

// $\mathsf{sqn} \in \{0,1\}^{32}, \mathsf{ctr} \in \{0,1\}^n$

1 : $m' \leftarrow\!\$\ \mathsf{Pad}_{\mathsf{SSH}}(m)$
2 : $m' \leftarrow \mathsf{len}(m') \| m'$
3 : $c \leftarrow \mathsf{CTR.Enc}(\mathsf{k}_e, \mathsf{ctr}, m')$
4 : $A \leftarrow \mathsf{sqn}$
5 : $\tau \leftarrow \mathsf{MAC}(\mathsf{k}_a, A \| m')$
6 : $\mathsf{sqn}' \leftarrow \mathsf{sqn} + 1$
7 : $\mathsf{ctr}' \leftarrow \mathsf{ctr} + \mathsf{len}(m')/B$
8 : **return** $(\varepsilon, c \| \tau, (\mathsf{sqn}', \mathsf{ctr}'))$

$\mathsf{SF.Enc}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}Stream}}((\mathsf{k}_e, \mathsf{k}_a), (\mathsf{sqn}, \mathsf{pos}), m)$

// $\mathsf{sqn} \in \{0,1\}^{32}, \mathsf{pos} \in \mathbb{N}_0$

1 : $m' \leftarrow\!\$\ \mathsf{Pad}_{\mathsf{SSH}}(m)$
2 : $m' \leftarrow \mathsf{len}(m') \| m'$
3 : $c \leftarrow \mathsf{Stream.Enc}(\mathsf{k}_e, \mathsf{pos}, m')$
4 : $A \leftarrow \mathsf{sqn}$
5 : $\tau \leftarrow \mathsf{MAC}(\mathsf{k}_a, A \| m')$
6 : $\mathsf{sqn}' \leftarrow \mathsf{sqn} + 1$
7 : $\mathsf{pos}' \leftarrow \mathsf{pos} + \mathsf{len}(m')$
8 : **return** $(\varepsilon, c \| \tau, (\mathsf{sqn}', \mathsf{pos}'))$

Fig. 4. Stateful encryption schemes using a generic EaM construction in SSH [32]. For block ciphers in CBC mode, the IV becomes the secure state because SSH uses IV chaining. For block ciphers in CTR mode, the counter counts the input blocks of the underlying block cipher and therefore becomes secure state. For stream ciphers, the channel state includes the internal cipher state of the stream cipher, that is, the position in the keystream. The only associated data is the sequence number sqn as the adversary-chosen state component, so the header H is empty.

dropped to avoid the bias in these bytes. All of these modes can be used in EaM mode [32, Sec. 6.3], with only minor adjustments to account for the different channel state. In our formal description, the latter two modes can be described by initializing pos to 1536 rather than zero, effectively skipping the first 1536 bytes. Although RC4 as a stream cipher does not require padding, the $\mathsf{Pad}_{\mathsf{SSH}}(m)$ function still pads m' to a multiple of 8 bytes, as mandated by the SSH specification [32, Sec. 6]. The stream cipher is never re-initialized for the lifetime of the key material, meaning that the current keystream position pos becomes part of the channel state.

4.3 Encrypt-then-MAC

SSH implementations widely support EtM as a vendor extension by OpenSSH. Here, the packet length is no longer encrypted but is instead, together with the sequence number, authenticated as part of the MAC input. The structure of this encryption mode is illustrated in Fig. 5, and the pseudocode translation from stateful encryption to AEAD in Fig. 6. The plaintext m' to be encrypted consists of the padding length field, the payload m, and the random padding.

The MAC τ is computed over the concatenation of the sequence number sqn, the packet length $\mathsf{len}(m')$, and the ciphertext c. These modifications are briefly described in [40, Sec. 1.5], while other aspects remain unchanged. For example, IV chaining is used in CBC mode and the counter is never reset in CTR mode. Thus, the channel state for each mode is identical to that of its EaM counterpart.

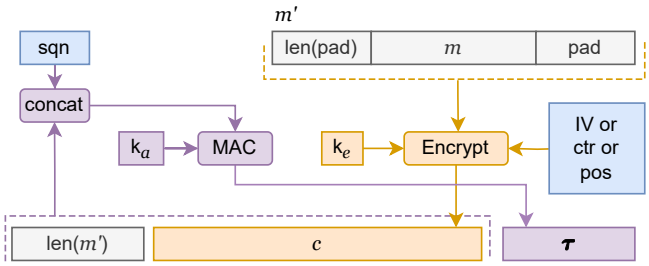

Fig. 5. Encryption of a binary packet in SSH, using generic Encrypt-then-MAC [40].

4.4 Galois/Counter Mode

According to RFC 5647 [18], in SSH the AEAD mode GCM requires a secret key k, a nonce N (which defines the start value of the internal counter), associated data A, and plaintext m. According to the general SSH rules, the plaintext must be padded to a multiple of the block length of AES. The output is a sequence of ciphertext blocks c and an authentication tag τ. The nonce N consists

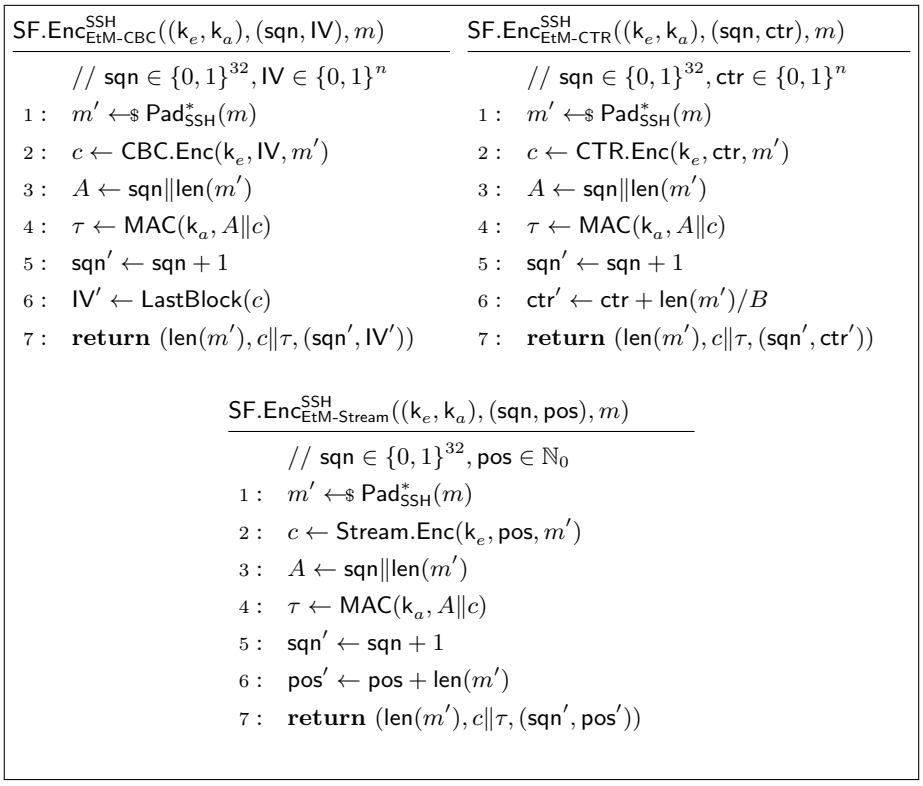

Fig. 6. Stateful encryption schemes using a generic EtM construction in SSH [40]. For block ciphers in CBC mode, the IV is part of the channel state because SSH uses IV chaining. For block ciphers in CTR mode, the counter counts the input blocks of the underlying block cipher and is therefore part of the channel state. For stream ciphers, the channel state includes the internal cipher state of the stream cipher, that is, the position in the keystream.

of 4 fixed bytes salt computed during key derivation and 8 bytes for the invocation counter ictr; the remaining 4 bytes are reserved for the block counter, which starts at 1 (Fig. 8, Fig. 7 line 2). A includes H, i.e., the packet length of the binary packet. In this mode, the packet length is not encrypted, see Fig. 8. The channel state includes the implicit sequence numbers; however, SSH with GCM does not use the sequence numbers in any cryptographic computation (Figs. 7 and 8); they are still incremented to allow their use with another AEAD mode after a possible key re-exchange. Instead, SSH uses the 8-byte invocation counter ictr that is incremented after each encryption of a packet.

$\mathsf{SF.Enc}^{\mathsf{SSH}}_{\mathsf{GCM}}((\mathsf{k}, \mathsf{salt}), (\mathsf{sqn}, \mathsf{ictr}), m)$

// $\mathsf{salt}, \mathsf{sqn} \in \{0,1\}^{32}, \mathsf{ictr} \in \{0,1\}^{64}$

1 : $m' \leftarrow\!\!\$\ \mathsf{Pad}^{*}_{\mathsf{SSH}}(m)$

2 : $N \leftarrow \mathsf{salt} \| \mathsf{ictr}$

3 : $A \leftarrow \mathsf{len}(m')$

4 : $(c, \tau) \leftarrow \mathsf{GCM.Enc}(\mathsf{k}, N, A, m')$

5 : $\mathsf{ictr}' \leftarrow \mathsf{ictr} + 1$

6 : $\mathsf{sqn}' \leftarrow \mathsf{sqn} + 1$

7 : **return** $(\mathsf{len}(m'), c \| \tau, (\mathsf{sqn}', \mathsf{ictr}'))$

Fig. 7. Stateful encryption scheme using GCM in SSH [18,35]. The sequence number is maintained but not used, denoted in gray color.

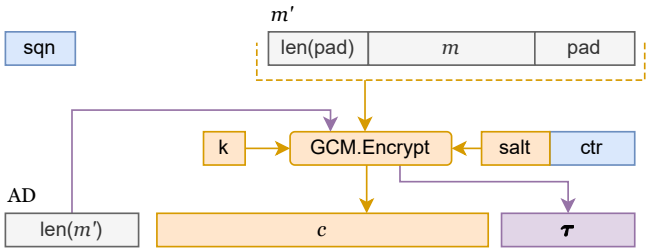

Fig. 8. Encryption of a binary packet in SSH using GCM [18,35].

4.5 ChaCha20-Poly1305

ChaCha20-Poly1305 [39] is a combination of the ChaCha20 stream cipher (re-initialized for each binary packet) and the MAC Poly1305. ChaCha20, like its

predecessor Salsa20, generates a keystream consisting of 512-bit blocks from a 4×4 matrix of 32-bit words using add-XOR-rotate operations. To produce one block of keystream, the ChaCha20 matrix is initialized with a 256-bit key k_e, a 128-bit constant, a 64-bit nonce N, and a 64-bit block counter ctr initialized to 0. An IETF [39] variant with a 96-bit nonce and a 32-bit block counter exists, but is not implemented by SSH.

In an attempt to reconcile the SSH BPP specification [32] with the attack presented in [2], the design of ChaCha20-Poly1305 for SSH incorporates a separately keyed ChaCha20 instance to encrypt the packet length field. This difference is illustrated in Fig. 10. The left-zero-padded 4-byte SSH sequence number is used directly as a nonce.

ChaCha20 is first initialized with k_2 to produce a single 512-bit block, of which 4 bytes are used to encrypt the length field via an XOR operation. Since a different keystream is used here, this mitigates the attack from [2], while also allowing the packet length to be encrypted. In the second step, ChaCha20 is used in combination with Poly1305 as an AEAD cipher. The previously encrypted 4-byte length field is used as associated data A.

$\mathsf{SF.Enc}^{\mathsf{SSH}}_{\mathsf{CCP}}((\mathsf{k}_2, \mathsf{k}_1), (\mathsf{sqn}, \varepsilon), m)$

// $\mathsf{sqn} \in \{0,1\}^{32}$

1 : $m' \leftarrow\!\$\ \mathsf{Pad}^{*}_{\mathsf{SSH}}(m)$

2 : $N \leftarrow 0^{32} \| \mathsf{sqn}$

3 : $c_1 \leftarrow \mathsf{ChaCha20.Enc}(\mathsf{k}_2, N, \mathsf{len}(m'))$

4 : $A \leftarrow c_1$

5 : $(c_2, \tau) \leftarrow \mathsf{ChaChaPoly.Enc}(\mathsf{k}_1, N, A, m')$

6 : $\mathsf{sqn}' \leftarrow \mathsf{sqn} + 1$

7 : **return** $(\varepsilon, c_1 \| c_2 \| \tau, \mathsf{sqn}')$

Fig. 9. Stateful encryption scheme using ChaCha20-Poly1305 in SSH. A separately keyed ChaCha20 instance encrypts the packet length to comply with RFC 4253 [32], distinguishing it from the EtM and GCM schemes.

5 Security Analysis

All models for secure channels from related work (Sections 3.3 and 3.4) provide the adversary with chosen-plaintext privileges. However, Terrapin is not a CPA attack—with chosen-plaintext privileges, schemes secure against Terrapin become vulnerable. We therefore analyze the security of the SSH stateful encryption schemes in three slightly different security models, which are selected by choosing one of the three oracles SND-CO, SND-KPA, SND-CPA.

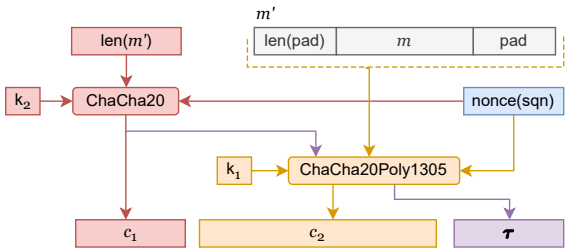

Fig. 10. Encryption of one binary packet using ChaCha20-Poly1305 in SSH [37]. The construction requires a 512-bit key that is split into two 256-bit keys k_1 and k_2.

The weakest security model, and the one modeling the Terrapin attack, is the one using SND-CO. If an AEAD scheme is insecure in this model, it is automatically insecure in the two stronger models. This is the case for the three EtM variants and for ChaCha20-Poly1305. Therefore, proving insecurity in the weakest model for these schemes as done in Sections 5.1 and 5.2 is sufficient.

The strongest of our security models is the CPA model. If we can prove that a cipher is secure in this model, then it is automatically secure in the two weaker models. This is the case for GCM, and the proof in Section 5.3 is straightforward: GCM does not use sequence numbers at all but instead *invocation counters* which are (likely) perfectly aligned with our bookkeeping indices j_s, j_r.

The most interesting class of stateful encryption schemes is EaM. Here, it is relatively straightforward to show that EaM-CTR and EaM-Stream are insecure in the KPA setting (Section 5.4), which also implies insecurity in the CPA setting. A proof of their security in the CO setting, which depends on the security of the PRG used to generate the keystream, can be found in Section 5.5. As RC4's PRG may be distinguishable from a random one [33], all concrete instantiations of EaM-Stream remain unproven. This leaves us with EaM-CBC, the most interesting mode. In Section 5.6, we show that this mode is *not* CINT-PCSA-CPA-secure. The attack methodology is similar to the BEAST attack on TLS [46], but has a different goal: instead of breaking confidentiality, we attempt to control $\mathsf{st}_r.\mathsf{IV}$ of the channel state. Finally, in Section 5.7 we show that EaM-CBC is CINT-PCSA-KPA-secure: breaking KPA security is equivalent to either forging a MAC or inverting the block cipher without the key.

5.1 Insecurity of ChaCha20-Poly1305 under CO

In the security game described in Fig. 2, stateful encryption schemes whose secure state is empty are vulnerable to a generic attack if their adversary-chosen state is predictable. In the context of SSH, the attack described below applies to ChaCha20-Poly1305 (cf. Table 2).

Definition 4 (Deterministic adversary-chosen state). *Let* SF *be a stateful encryption scheme with state* $\mathsf{st}_i = (\xi_i, \eta_i)$ *after the* i*-th invocation of* SF.Enc. *We*

say that SF *has a* deterministic adversary-chosen state *if there exists a publicly known deterministic polynomial-time function* $f_{\mathsf{adv}} : \mathcal{S}_{\mathsf{adv}} \to \mathcal{S}_{\mathsf{adv}}$ *such that, for every call to* $\mathsf{SF.Enc}$ *and every* $i \geq 0$, $\xi_{i+1} = f_{\mathsf{adv}}(\xi_i)$. *Thus, given* ξ_i, *an adversary can efficiently determine all subsequent adversary-chosen states without the need to set them first.*

Theorem 1. *Let* SF *be a stateful encryption scheme where the adversary-chosen state* ξ *is deterministic and* $\eta = \varepsilon$. *Then* SF *is* not CINT-PCSA-CO*-secure.*

Proof. We construct a PPT adversary $\mathcal{A}$ against the CINT-PCSA-CO game of Fig. 2 that proceeds as follows:

1. $\mathcal{A}$ samples $\xi \leftarrow\!\!\$\ \mathcal{S}_{\mathsf{adv}}$ and invokes $\mathsf{SET}_r(\xi)$ and $\mathsf{SET}_s(\xi)$, thereby the initial state is $\xi_0 = \xi$.
2. $\mathcal{A}$ queries $\mathsf{SND\text{-}CO}(\ell)$ twice for some length $\ell \geq 1$. The challenger samples and stores two messages $M[0]$ and $M[1]$, thereby advancing st_s twice, and returns (H_1, c_1) and (H_2, c_2) to $\mathcal{A}$. As both messages are independent and uniformly distributed, $M[0] \neq M[1]$ except with probability $2^{-8\ell}$.
3. $\mathcal{A}$ calls $\mathsf{SET}_r(f_{\mathsf{adv}}(\xi))$; this is permitted as RCV has not yet been called. As ξ is deterministic, f_{adv} exists and is known to the adversary.
4. $\mathcal{A}$ calls $\mathsf{RCV}(\mathrm{H}_2, c_2)$. Inside $\mathsf{SF.Dec}$, the receiver state st_r matches the corresponding sender state st_s used to encrypt c_2, as $\eta = \varepsilon$ and ξ was deliberately set in the previous step. Consequently, c_2 decrypts to $M[1]$. Since $j_r = 0$, the oracle compares the result against $M[0]$; as $M[1] \neq M[0]$, it sets $res = M[1] \neq \perp$.
5. $\mathcal{A}$ outputs $b' = 1$ if RCV returns a value other than $\perp$, and $b' = 0$ otherwise. For $b = 1$ this yields $b' = 1$ whenever $M[0] \neq M[1]$, i.e. with probability $1 - 2^{-8\ell}$; for $b = 0$ the oracle always returns $\perp$, so $b' = 0$ with certainty. Hence, $\mathsf{Adv}^{\text{CINT-PCSA-CO}}_{\mathcal{A}}(\lambda) \geq \frac{1}{2} - 2^{-(8\ell+1)}$ is non-negligible.

Corollary 1. $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{CCP}}$ *is not* CINT-PCSA-CO*-secure.*

Proof. By Table 2, the scheme's adversary-chosen state consists solely of the sequence number ($\xi = \mathsf{sqn}$) and the secure state is empty ($\eta = \varepsilon$). The sequence number is deterministic by $f_{\mathsf{adv}}(\xi) \equiv \xi + 1 \bmod 2^{32}$ and the corollary follows directly from Theorem 1.

5.2 Insecurity of Encrypt-then-MAC under CO

For EtM modes, the state is the adversary-chosen sequence number sqn together with additional secure state: a chained IV (CBC), a counter (CTR), or a keystream position (Stream). As the secure state is no longer empty, the generic attack of Section 5.1 does not apply directly. The same attack nonetheless breaks CINT-PCSA-CO security. As shown in Fig. 6, the tag is computed over the sequence number, the packet length, and the ciphertext—that is, over $\mathsf{sqn} \| \mathsf{len}(m') \| c$; the secure state never enters the MAC. Leaving the ciphertext unchanged and only shifting sqn via SET_r therefore keeps the MAC valid, so

the adversary from Section 5.1 applies without change. The secure state is used only by the cipher, affecting only the decrypted plaintext. It is updated by the cipher—using the previous ciphertext block in CBC, the number of blocks in CTR, or the number of bytes in Stream—so it depends on the earlier traffic, not on sqn, which only counts packets. The two are therefore independent. If this secure state does not match, the decrypted plaintext is wrong, but the tag still verifies since it covers the ciphertext without the secure state. While such a plaintext may be rejected at the application layer [5], we exclude this consideration from our model (cf. Section 3.2).

Theorem 2. $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EtM\text{-}CBC}}$, $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EtM\text{-}CTR}}$, *and* $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EtM\text{-}Stream}}$ *are not* CINT-PCSA-CO*-secure.*

Proof. We reuse the adversary $\mathcal{A}$ from the proof of Theorem 1, excluding the first step, against the CINT-PCSA-CO game of Fig. 2 instantiated with one of the EtM schemes of Fig. 6: $\mathcal{A}$ queries $\mathsf{SND\text{-}CO}(\ell)$ twice, calls $\mathsf{SET}_r(1)$, queries $\mathsf{RCV}(\mathrm{H}_2, c_2)$, and decides as before. Its analysis rests on two properties, both of which still hold. The argument is identical for all three modes, which differ only in the additional state.

(i) Verification succeeds. The ciphertext c_2 was produced under $\mathsf{st}_s.\mathsf{sqn} = 1$, so its tag is $\mathsf{MAC}(\mathsf{k}_a, 1 \| \mathsf{len}(m'_2) \| c_2)$. As $\mathsf{SET}_r(1)$ sets $\mathsf{st}_r.\mathsf{sqn} = 1$ and $\mathcal{A}$ forwards (H_2, c_2) unchanged, $\mathsf{SF.Dec}$ recomputes the tag over the identical input, so verification succeeds and returns a plaintext rather than $\perp$. The additional state does not enter the MAC, so this is independent of its value.

(ii) The accepted plaintext is fresh. Since SET_r shifts only $\mathsf{st}_r.\mathsf{sqn}$, the receiver decrypts c_2 under the additional state from $\mathsf{SF.Init}$ rather than the one used to encrypt it, recovering some $m^* \neq M[1]$ in general. Decryption of a fixed ciphertext is a bijection, so m^* is uniform and independent of $M[0]$, since $M[1]$ was sampled freshly. With $j_r = 0$, the RCV oracle thus sets $res = m^* \neq \perp$ unless $m^* = M[0]$, which occurs with probability $2^{-8\ell}$.

Properties (i) and (ii) are what the proof of Theorem 1 requires, so it carries over and $\mathsf{Adv}^{\text{CINT-PCSA-CO}}_{\mathcal{A}}(\lambda) \geq \frac{1}{2} - 2^{-(8\ell+1)}$ is non-negligible.

5.3 Security of GCM under CPA

In SSH's GCM mode (Fig. 7), the sequence number is unused: the nonce is $N = \mathsf{salt} \| \mathsf{ictr}$ and the associated data is the packet length; sqn enters neither of them. Consequently, the SET oracles, which write only sqn, cannot influence the channel, and GCM resists sequence number manipulation. This matches the finding of [5] that GCM is unaffected by the Terrapin attack.

Security therefore rests entirely on the invocation counter ictr. We prove it under the assumption that ictr is never reused, is *independent of* sqn and aligned with the bookkeeping indices up to a fixed offset δ, that is, $\mathsf{ictr}_s = j_s + \delta$ and $\mathsf{ictr}_r = j_r + \delta$ with ictr_r advancing only on accepted packets. The independence assumption is essential. If an implementation instead computed the invocation counter from the sequence number—say $\mathsf{ictr} = \mathsf{ictr}_0 + \mathsf{sqn}$—then ictr would be

a function of sqn, collapsing the state to sqn. The generic attack of Section 5.1 then applies, and the channel would *not* be CINT-PCSA-CO-secure.

Theorem 3. *Under the* ictr *assumption above,* $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{GCM}}$ *is* CINT-PCSA-CPA*-secure, with*

$$\mathsf{Adv}^{\text{CINT-PCSA-CPA}}_{\mathcal{A}}(\lambda) \leq \epsilon^{\text{INT-CTXT}}_{\mathsf{GCM}}.$$

Proof (Sketch). **Game** G_0 is the original CINT-PCSA-CPA game. By assumption the effective state on each side is the invocation counter, $\mathsf{st}_s = \mathsf{ictr}_s = j_s + \delta$ and $\mathsf{st}_r = \mathsf{ictr}_r = j_r + \delta$. Since the nonce is $N = \mathsf{salt}\|\mathsf{ictr}$ and sqn enters neither nonce nor associated data, the SET oracles—which write only sqn—leave $\mathsf{SND}/\mathsf{RCV}$ unaffected. The packet sent at index i uses nonce $\mathsf{salt}\|(i+\delta)$; these nonces are pairwise distinct in i.

Game G_1 aborts whenever RCV accepts a tuple (H, c) for which the GCM triple $(N, A, c\|\tau)$ reconstructed by $\mathsf{SF.Dec}$—with nonce $N = \mathsf{salt}\|(j_r + \delta)$ from the receiver's state and associated data $A = \mathrm{H}$ from the adversary's input—was never produced by an SND query. Accepting such a fresh triple constitutes a GCM ciphertext forgery, so $|\Pr[G_0] - \Pr[G_1]| \leq \epsilon^{\text{INT-CTXT}}_{\mathsf{GCM}}$. Hence, from G_1 on, RCV returns $\perp$ on every query whose triple was not produced by an SND query.

Game G_2 records, for each SND output, the pair (H, c) and the index j at which it was created; this is bookkeeping only, so $\Pr[G_2] = \Pr[G_1]$. Consider a query $\mathsf{RCV}(\mathrm{H}, c)$. The receiver reconstructs its triple under nonce $\mathsf{salt}\|(j_r + \delta)$; this is unique because each invocation counter is used only once; by G_1 the query is rejected unless (H, c) equals that SND output; in this case decryption returns the recorded message $M[j_r]$, and since $m = M[j_r]$, the RCV oracle sets $res = \perp$. In both cases, RCV returns $\perp$ regardless of the challenge bit, so $\mathcal{A}$ has advantage 0 in G_2.

5.4 Insecurity of EaM-CTR/-Stream under KPA

For EaM, the tag is $\mathsf{MAC}(\mathsf{k}_a, \mathsf{sqn}\|m')$ (Fig. 4). For CTR and Stream variants, where $c = m' \oplus ks$, this breaks KPA integrity. An adversary that learns the plaintext m' also learns the keystream $ks = c \oplus m'$, and can therefore re-encrypt any plaintext for which it holds a valid tag under the receiver's keystream; SET_r then aligns the sequence number to match that tag, and the receiver accepts an out-of-order message. The attack requires KPA rather than CO, since recovering ks needs the plaintext, and it is specific to these modes.

Theorem 4. $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}CTR}}$ *and* $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}Stream}}$ *are not* CINT-PCSA-KPA*-secure.*

Proof. We construct a PPT adversary $\mathcal{A}$ that proceeds as follows:

1. $\mathcal{A}$ queries $\mathsf{SND\text{-}KPA}(2B - 9)$ twice.[2] The oracle samples random m_1, m_2 and returns $(\varepsilon, c_i\|\tau_i, m_i)$, so $\mathcal{A}$ learns both plaintext and ciphertext; they

[2] The term $2B - 9$ ensures minimal 4-byte padding even if $B = 8$ in the case of EaM-Stream, considering that there are 5 bytes of length fields.

differ except with probability $2^{-8(2B-9)}$. This yields an encryption input with minimal padding:

$$m'_i = (2B-4)\|\mathsf{04}\|m_{i,1}\cdots m_{i,2B-9}\|pad_{i,1}\cdots pad_{i,4},$$

where $\tau_i = \mathsf{MAC}(\mathsf{k}_a, (i-1)\|m'_i)$ and $c_i = m'_i \oplus ks_i$, with ks_i the i-th packet's keystream. From m'_1, $\mathcal{A}$ recovers $ks_1 = c_1 \oplus m'_1$ on all but the four padding bytes and keeps the valid MAC pair $(1\|m'_2, \tau_2)$; c_2 is unused. Now $M[0] = m_1$ and $M[1] = m_2$.

2. $\mathcal{A}$ calls $\mathsf{SET}_r(1)$, setting $\mathsf{st}_r.\mathsf{sqn} = 1$; this is admissible, as $p_r = 0$. Since SET_r only writes sqn, the receiver's keystream is unchanged, so the first RCV still uses ks_1.
3. $\mathcal{A}$ calls $\mathsf{RCV}(\varepsilon, c\|\tau_2)$ with $c = m'_2 \oplus ks_1$. The oracle decrypts $c \oplus ks_1 = m'_2$ and checks $\mathsf{MAC}(\mathsf{k}_a, \mathsf{sqn}\|m'_2)$; as $\mathsf{st}_r.\mathsf{sqn} = 1$ this equals τ_2, so $\mathsf{SF.Dec}$ returns m_2. With $j_r = 0$ the oracle compares m_2 against $M[0] = m_1$, and since $m_2 \neq m_1$ it sets $res = m_2 \neq \perp$.
4. $\mathcal{A}$ outputs $b' = 1$ if RCV returns a value other than $\perp$, and $b' = 0$ otherwise.

$\mathcal{A}$ knows neither ks_1 nor the four padding bytes of m'_2, so the receiver recovers m'_2 and τ_2 verifies only with probability 2^{-32}, giving $\mathsf{Adv}^{\text{CINT-PCSA-KPA}}_{\mathcal{A}}(\lambda) \approx 2^{-33}$, which is non-negligible.

5.5 Security of EaM-CTR/-Stream under CO

Switching from $\mathsf{SND\text{-}KPA}$ to $\mathsf{SND\text{-}CO}$ removes the known payload that enabled the attack of Section 5.4, leaving only structural fields known. The adversary can compute the first five bytes of m' from the byte length $\mathsf{len}(m)$ it selects via $\mathsf{SND\text{-}CO}$: the packet length equals $\mathsf{len}(m) + n + 1$ and the padding length is n, where $n \geq 4$ with $\mathsf{len}(m)+n+5 \equiv 0 \pmod{B}$ and n minimal. The adversary thus learns the keystream on five bytes of every packet, separated by the remaining $B-5$ random bytes of m and *pad*. As $B = \Omega(\lambda)$ in our model, this unknown gap grows with λ.

Both modes' encryption functions are structurally equivalent XOR stream ciphers: to encrypt m', they compute a keystream ks using a PRG and then compute $c = m' \oplus ks$. The MAC is computed over m', so a forgery must reproduce a sent plaintext exactly. For the following proof, we assume that for CTR the counter value is never reused, and for Stream the internal state of the keystream generator never repeats. In SSH, this is achieved by regularly rotating the session key via a new key exchange.

Theorem 5. *Let* SF *be the EaM stateful encryption scheme over an XOR stream cipher* Enc *and a MAC* MAC*. Then, for every PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\text{CINT-PCSA-CO}}_{\mathcal{A}}(\lambda) \leq \epsilon^{\text{SUF-CMA}}_{\mathsf{MAC}} + \epsilon^{\mathsf{PRG}}_{\mathsf{Enc}} + 2^{-8(B-5)}.$$

Proof (Sketch). We argue over a sequence of games; w.l.o.g. packets are minimal and span one B-block, of which the adversary knows the five structural bytes.

Game G_0 is the original CINT-PCSA-CO game.

Game G_1 aborts if RCV accepts a pair $(\mathsf{sqn}_r \| m^*, \tau)$ that MAC never produced during any SND query, giving $|\Pr[G_0] - \Pr[G_1]| \leq \epsilon^{\text{SUF-CMA}}_{\mathsf{MAC}}$.

Game G_2 replaces the keystream PRG with a uniformly random keystream, so $|\Pr[G_1] - \Pr[G_2]| \leq \epsilon^{\text{PRG}}_{\mathsf{Enc}}$.

In G_2 the keystream is uniform. To make RCV return a value other than $\perp$, the adversary must submit c with $c \oplus ks_r = m'_j$ under the receiver's keystream block ks_r. It knows ks_r on at most the five structural bytes (revealed by the packet sent under that state); the remaining $B-5$ bytes are uniform and unseen, so the single forgery attempt that the adversary can make before the channel terminates succeeds with probability at most $2^{-8(B-5)}$.

For CTR, the PRG is a block cipher in counter mode, which is a PRG whenever the cipher is a Pseudorandom Permutation (PRP); the term $\epsilon^{\mathsf{PRG}}_{\mathsf{Enc}}$ reduces to the cipher's PRP advantage (up to the usual birthday bound) and is negligible, so EaM-CTR is CINT-PCSA-CO-secure.

For Stream mode with RC4, the same reduction would apply but it is unclear whether the premise holds: RC4's keystream has positional biases [3], so $\epsilon^{\mathsf{PRG}}_{\mathsf{RC4}}$ may be non-negligible.

5.6 Insecurity of EaM-CBC under CPA

EaM with CBC is not CINT-PCSA-CPA-secure: an adaptive chosen-plaintext adversary can mount a probabilistic forgery in the style of the BEAST attack against TLS.

Theorem 6. $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}CBC}}$ *is not* CINT-PCSA-CPA*-secure.*

Proof. Let *pad* denote the four random padding bytes of message 2 below, which are unknown to the adversary, and let pad^* be the adversary's guess for them. Write $T = \texttt{0xf2}^{B-4} \| pad^*$. The following sequence causes the SND oracle to emit three messages of two ciphertext blocks each; the first two carry 4 padding bytes, the third B padding bytes:

$$\begin{aligned}
(\varepsilon, cb_0 \| cb_1 \| \tau_0) &\leftarrow \mathsf{SND}(\texttt{0xf0}^{B-5} \| \texttt{0xf1}^{B-4}),\\
(\varepsilon, cb_2 \| cb_3 \| \tau_1) &\leftarrow \mathsf{SND}(\texttt{0xf0}^{B-5} \| \texttt{0xf2}^{B-4}),\\
(\varepsilon, cb_4 \| cb_5 \| \tau_2) &\leftarrow \mathsf{SND}\big((T \oplus cb_3 \oplus cb_0)_{5\ldots B-1}\big),\\
&\quad\ \mathsf{SET}_r(1),\\
m^* &\leftarrow \mathsf{RCV}(\varepsilon, cb_0 \| cb_4 \| \tau_1).
\end{aligned}$$

The first two messages have the same length and the same $\texttt{0xf0}^{B-5}$ prefix, so their first plaintext block is identical; call it P_0. Their second plaintext blocks are $\texttt{0xf1}^{B-4} \| pad'$ and $\texttt{0xf2}^{B-4} \| pad$, respectively. Since CBC chains the IV, cb_0 is P_0 encrypted under the initial IV, and message 3 is encrypted under cb_3. Its first plaintext block is $P_1 = len \| padlen \| m_3$, where $len \| padlen$ is the 5-byte header added by SND and $m_3 = (T \oplus cb_3 \oplus cb_0)_{5\ldots B-1}$ are the $B-5$ payload bytes chosen by the adversary; the second block cb_5 is not used.

At the receiver, the IV is still the initial one (nothing has been received), and $\mathsf{SET}_r(1)$ sets its sequence number to 1, the index under which message 2 and its tag τ_1 were sent. The receiver decrypts $cb_0 \| cb_4$ block by block. The first block cb_0 decrypts back to P_0, which is also the first block of message 2. The second block decrypts to $\mathsf{Dec}(cb_4) \oplus cb_0 = P_1 \oplus cb_3 \oplus cb_0$, where Dec is the block-cipher inverse. By the choice of m_3,

$$\mathsf{Dec}(cb_4) \oplus cb_0 = \big(len \| padlen \oplus (cb_3 \oplus cb_0)_{0\ldots4}\big) \| T_{5\ldots B-1}.$$

The tag τ_1 verifies exactly when this block equals message 2's second block $\texttt{0xf2}^{B-4} \| pad$. Bytes $5, \ldots, B-1$ agree iff the guess pad^* matches pad on the positions it covers; bytes $0, \ldots, 4$ agree iff the fixed header $len \| padlen$ equals $(\texttt{0xf2}^{B-4} \| pad \oplus cb_3 \oplus cb_0)_{0\ldots4}$, an event over the block-cipher outputs cb_0 and cb_3. Altogether, $\min(9, B)$ bytes are outside the adversary's control, so the forgery succeeds with probability 2^{-72} for $B \geq 9$ and 2^{-64} for $B = 8$, where byte 4 of the target block is pad_0 rather than $\texttt{0xf2}$. On success, RCV returns message 2's plaintext, which differs from message 1 stored in $M[0]$, so the receiver accepts a different message at position 0. The adversary outputs $b' = 1$ if RCV returns a value other than $\perp$ and $b' = 0$ otherwise; as a non-$\perp$ answer occurs only when $b = 1$ and the forgery succeeds, $\mathsf{Adv}^{\text{CINT-PCSA-CPA}}_{\mathcal{A}}(\lambda) \geq 2^{-73}$ for every $B \geq 8$.

The probability 2^{-72} does not yield a practically exploitable attack. Each attempt transfers 8 blocks, so achieving probable success requires $8 \cdot 2^{72} = 2^{75}$ blocks—more than the $\approx 2^{64}$ birthday bound of a 128-bit cipher, so in that case a collision attack is cheaper. However, our attack's cost is fixed in the block length n_{Enc}, while a collision attack costs $\approx 2^{n_{\mathsf{Enc}}/2}$: for $n_{\mathsf{Enc}} \geq 150$ the comparison flips. SSH rekeying has the same effect. With a rekey interval of l blocks, the per-key collision probability is only $l(l-1)/2$, requiring $\approx 2/l$ blocks: about 2^{97} for $l = 2^{32}$, the interval recommended by [38].

5.7 Security of EaM-CBC under KPA

For the following proof, we assume the number of SND invocations is bounded such that birthday collisions on the block cipher are negligible. In SSH, this is achieved by timely rekeying of the session key.

Theorem 7. $\mathsf{SF}^{\mathsf{SSH}}_{\mathsf{EaM\text{-}CBC}}$ *is* CINT-PCSA-KPA*-secure. For every PPT adversary, the advantage is*

$$\mathsf{Adv}^{\text{CINT-PCSA-KPA}}_{\mathcal{A}}(\lambda) \leq \epsilon^{\text{SUF-CMA}}_{\mathsf{MAC}} + \epsilon^{\text{SPRP}}_{\mathsf{Enc}} + 2^{-8B}.$$

Proof (Sketch). We argue over a sequence of games; w.l.o.g. packets are minimal and span one B-block.

Game G_0 is the original CINT-PCSA-KPA game.

Game G_1 aborts if RCV accepts a pair $(\mathsf{sqn}_r \| m', \tau)$ that MAC never produced during an SND query, giving $|\Pr[G_0] - \Pr[G_1]| \leq \epsilon^{\text{SUF-CMA}}_{\mathsf{MAC}}$.

Game G_2 replaces the block cipher encryption Enc with a random permutation π, so $|\Pr[G_1]-\Pr[G_2]| \leq \epsilon^{\mathrm{SPRP}}_{\mathsf{Enc}}$. In G_2 the permutation is uniform. IV chaining couples the receiver state—an accepted RCV advances $\mathsf{st}_r.\mathsf{sqn}$, $\mathsf{st}_r.\mathsf{IV}$, and j_r in lockstep—and the only oracle that moves $\mathsf{st}_r.\mathsf{sqn}$ on its own is SET_r, which is admissible only before the first RCV, when $\mathsf{st}_r.\mathsf{IV}$ is still the secret initial $\mathsf{IV}_{\mathsf{init}}$. If the adversary advances the receiver by delivering genuine packets—learning $\mathsf{st}_r.\mathsf{IV}$ via chaining—then by G_1 any accepted RCV must decrypt to the unique m'_k tagged under $\mathsf{st}_r.\mathsf{sqn}$. As CBC decryption under a fixed IV is a bijection, the submitted ciphertext must be the matching SND output, so stateful decryption returns $M[j_r]$ and gives $res = \perp$; a known IV is useless. To reach a fresh target, the adversary must therefore use SET_r, possible only before the first RCV, where $\mathsf{st}_r.\mathsf{IV} = \mathsf{IV}_{\mathsf{init}}$; the fail-stop (f_r) then permits a single attempt. As only the first packet was encrypted under $\mathsf{IV}_{\mathsf{init}}$ and a fresh target has $k \neq 0$, the ciphertext $\pi(m'_k \oplus \mathsf{IV}_{\mathsf{init}})$ is a fresh output of π, matching with probability 2^{-8B}.

6 Summary and Outlook

In our extended theoretical model, we have shown that if a vulnerability corresponding to a SET oracle exists in an implementation, four out of eight stateful encryption schemes can be broken, while the security of one additional scheme depends on the security of RC4. In SSH without the strict KEX countermeasure, the protocol exhibits a real-world SET oracle, as shown in [5]. When considering Known Plaintext Attacks (KPAs) and Chosen Plaintext Attacks (CPAs) adversaries, all modes are insecure except GCM and, for KPA only, EaM-CBC. These results do not consider layers above the Binary Packet Protocol, which on the one hand implement additional validity checks but on the other hand have known-plaintext components and might even allow limited chosen plaintext. We leave a study of the KPA/CPA potentials of these layers as future work.

As for insecure modes, we note that not all attacks are equally effective. In particular, the KPA-insecurity of EaM-CTR and EaM-Stream, as well as the CPA-insecurity of EaM-CBC, depend on the minimum padding length, which is 4 bytes in SSH, independent of the block length. A modification to the SSH specification or its implementations to increase the minimum padding length could strengthen the security of these modes in our model. Finally, we cannot prove the security of RC4-based stream cipher modes in SSH mainly due to defects in RC4 itself. On the other hand, we cannot provide an attack against RC4, leaving the question of its security in SSH channels open.

There are many interesting constructions of secure channels that deserve a similar treatment as SSH. First, there are *standardized* channels like TLS, IPsec ESP/AH [24,23,43,10,11], DTLS [44,45,4,14], SRTP, and, perhaps most interestingly, QUIC [19,15]. Second, there are non-IETF protocols like tcpcrypt, MinimaLT, CurveCP, WireGuard, and OpenVPN (see [13] for references). Some of these protocols are widely deployed, but reliable empirical data is lacking. To better understand the security of these schemes, a thorough analysis of the stateful encryption layer (and the AKE protocol used) would be desirable.

References

1. Albrecht, M.R., Degabriele, J.P., Hansen, T.B., Paterson, K.G.: A surfeit of SSH cipher suites. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 1480–1491. ACM Press, Vienna, Austria (Oct 24–28, 2016). https://doi.org/10.1145/2976749.2978364
2. Albrecht, M.R., Paterson, K.G., Watson, G.J.: Plaintext recovery attacks against SSH. In: 2009 IEEE Symposium on Security and Privacy. pp. 16–26. IEEE Computer Society Press, Oakland, CA, USA (May 17–20, 2009). https://doi.org/10.1109/SP.2009.5
3. AlFardan, N.J., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.N.: On the security of RC4 in TLS. In: King, S.T. (ed.) USENIX Security 2013. pp. 305–320. USENIX Association, Washington, DC, USA (Aug 14–16, 2013), https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/alFardan
4. AlFardan, N.J., Paterson, K.G.: Lucky thirteen: Breaking the TLS and DTLS record protocols. In: 2013 IEEE Symposium on Security and Privacy. pp. 526–540. IEEE Computer Society Press, Berkeley, CA, USA (May 19–22, 2013). https://doi.org/10.1109/SP.2013.42
5. Bäumer, F., Brinkmann, M., Schwenk, J.: Terrapin attack: Breaking SSH channel integrity by sequence number manipulation. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association, Philadelphia, PA, USA (Aug 14–16, 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/b%C3%A4umer
6. Bellare, M., Kohno, T., Namprempre, C.: Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In: Atluri, V. (ed.) ACM CCS 2002. pp. 1–11. ACM Press, Washington, DC, USA (Nov 18–22, 2002). https://doi.org/10.1145/586110.586112
7. 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 Berlin Heidelberg, Germany, Kyoto, Japan (Dec 3–7, 2000). https://doi.org/10.1007/3-540-44448-3_41
8. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. Journal of Cryptology **21**(4), 469–491 (Oct 2008). https://doi.org/10.1007/s00145-008-9026-x
9. Boyd, C., Hale, B., Mjølsnes, S.F., Stebila, D.: From stateless to stateful: Generic authentication and authenticated encryption constructions with application to TLS. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 55–71. Springer, Cham, Switzerland, San Francisco, CA, USA (Feb 29 – Mar 4, 2016). https://doi.org/10.1007/978-3-319-29485-8_4
10. Degabriele, J.P., Paterson, K.G.: Attacking the IPsec standards in encryption-only configurations. In: 2007 IEEE Symposium on Security and Privacy. pp. 335–349. IEEE Computer Society Press, Oakland, CA, USA (May 20–23, 2007). https://doi.org/10.1109/SP.2007.8
11. Degabriele, J.P., Paterson, K.G.: On the (in)security of IPsec in MAC-then-encrypt configurations. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010. pp. 493–504. ACM Press, Chicago, Illinois, USA (Oct 4–8, 2010). https://doi.org/10.1145/1866307.1866363

12. Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Protzenko, J., Rastogi, A., Swamy, N., Zanella-Béguelin, S., Bhargavan, K., Pan, J., Zinzindohoue, J.K.: Implementing and proving the TLS 1.3 record layer. In: 2017 IEEE Symposium on Security and Privacy. pp. 463–482. IEEE Computer Society Press, San Jose, CA, USA (May 22–26, 2017). https://doi.org/10.1109/SP.2017.58
13. Enghardt, R., Pauly, T., Perkins, C., Rose, K., Wood, C.A.: A Survey of the Interaction between Security Protocols and Transport Services. RFC 8922 (Oct 2020). https://doi.org/10.17487/RFC8922, https://www.rfc-editor.org/info/rfc8922
14. Erinola, N., Maehren, M., Merget, R., Somorovsky, J., Schwenk, J.: Exploring the unknown DTLS universe: Analysis of the DTLS server ecosystem on the internet. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 4859–4876. USENIX Association, Anaheim, CA, USA (Aug 9–11, 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/erinola
15. Fischlin, M., Günther, F., Janson, C.: Robust channels: Handling unreliable networks in the record layers of QUIC and DTLS 1.3. Cryptology ePrint Archive, Report 2020/718 (2020), https://eprint.iacr.org/2020/718
16. Fischlin, M., Günther, F., Marson, G.A., Paterson, K.G.: Data is a stream: Security of stream-based channels. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 545–564. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015). https://doi.org/10.1007/978-3-662-48000-7_27
17. Harris, B.: Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol. RFC 4345 (Jan 2006). https://doi.org/10.17487/RFC4345, https://www.rfc-editor.org/info/rfc4345
18. Igoe, K., Solinas, J.: AES Galois Counter Mode for the Secure Shell Transport Layer Protocol. RFC 5647 (Aug 2009). https://doi.org/10.17487/RFC5647, https://www.rfc-editor.org/info/rfc5647
19. Iyengar, J., Thomson, M.: QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000 (May 2021). https://doi.org/10.17487/RFC9000, https://www.rfc-editor.org/info/rfc9000
20. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_17
21. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: Authenticated confidential channel establishment and the security of TLS-DHE. Journal of Cryptology **30**(4), 1276–1324 (Oct 2017). https://doi.org/10.1007/s00145-016-9248-2
22. Katz, J., Yung, M.: Unforgeable encryption and chosen ciphertext secure modes of operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer Berlin Heidelberg, Germany, New York, NY, USA (Apr 10–12, 2001). https://doi.org/10.1007/3-540-44706-7_20
23. Kent, S.: IP Authentication Header. RFC 4302 (Dec 2005). https://doi.org/10.17487/RFC4302, https://www.rfc-editor.org/info/rfc4302
24. Kent, S.: IP Encapsulating Security Payload (ESP). RFC 4303 (Dec 2005). https://doi.org/10.17487/RFC4303, https://www.rfc-editor.org/info/rfc4303
25. Kohno, T., Palacio, A., Black, J.: Building secure cryptographic transforms, or how to encrypt and MAC. Cryptology ePrint Archive, Report 2003/177 (2003), https://eprint.iacr.org/2003/177

26. 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 Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2001). https://doi.org/10.1007/3-540-44647-8_19
27. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40041-4_24
28. Lonvick, C.M., Lehtinen, S.: The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250 (Jan 2006). https://doi.org/10.17487/RFC4250, https://www.rfc-editor.org/info/rfc4250
29. Lonvick, C.M., Ylonen, T.: The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Jan 2006). https://doi.org/10.17487/RFC4252, https://www.rfc-editor.org/info/rfc4252
30. Lonvick, C.M., Ylonen, T.: The Secure Shell (SSH) Connection Protocol. RFC 4254 (Jan 2006). https://doi.org/10.17487/RFC4254, https://www.rfc-editor.org/info/rfc4254
31. Lonvick, C.M., Ylonen, T.: The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Jan 2006). https://doi.org/10.17487/RFC4251, https://www.rfc-editor.org/info/rfc4251
32. Lonvick, C.M., Ylonen, T.: The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Jan 2006). https://doi.org/10.17487/RFC4253, https://www.rfc-editor.org/info/rfc4253
33. Mantin, I., Shamir, A.: A practical attack on broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer Berlin Heidelberg, Germany, Yokohama, Japan (Apr 2–4, 2002). https://doi.org/10.1007/3-540-45473-X_13
34. McGrew, D.: An Interface and Algorithms for Authenticated Encryption. RFC 5116 (Jan 2008). https://doi.org/10.17487/RFC5116, https://www.rfc-editor.org/info/rfc5116
35. Miller, D.: Fixed AES-GCM modes for the SSH protocol. Internet-Draft draft-miller-sshm-aes-gcm-01, Internet Engineering Task Force (Nov 2025), https://datatracker.ietf.org/doc/draft-miller-sshm-aes-gcm/01/, work in Progress
36. Miller, D.: SSH Strict KEX extension. Internet-Draft draft-ietf-sshm-strict-kex-01, Internet Engineering Task Force (Nov 2025), https://datatracker.ietf.org/doc/draft-ietf-sshm-strict-kex/01/, work in Progress
37. Miller, D., Tatham, S., Josefsson, S.: Secure Shell (SSH) authenticated encryption cipher: chacha20-poly1305. Internet-Draft draft-ietf-sshm-chacha20-poly1305-04, Internet Engineering Task Force (May 2026), https://datatracker.ietf.org/doc/draft-ietf-sshm-chacha20-poly1305/04/, work in Progress
38. Namprempre, C., Kohno, T., Bellare, M.: The Secure Shell (SSH) Transport Layer Encryption Modes. RFC 4344 (Jan 2006). https://doi.org/10.17487/RFC4344, https://www.rfc-editor.org/info/rfc4344
39. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols. RFC 8439 (Jun 2018). https://doi.org/10.17487/RFC8439, https://www.rfc-editor.org/info/rfc8439
40. OpenSSH: This documents OpenSSH's deviations and extensions to the published SSH protocol. Specification, OpenSSH (Feb 2026), https://cvsweb.openbsd.org/annotate/src/usr.bin/ssh/PROTOCOL?rev=1.60
41. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the TLS record protocol. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 372–389. Springer Berlin Heidelberg,

Germany, Seoul, South Korea (Dec 4–8, 2011). https://doi.org/10.1007/978-3-642-25385-0_20
42. Paterson, K.G., Watson, G.J.: Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 345–361. Springer Berlin Heidelberg, Germany, French Riviera (May 30 – Jun 3, 2010). https://doi.org/10.1007/978-3-642-13190-5_18
43. Paterson, K.G., Yau, A.K.L.: Cryptography in theory and practice: The case of encryption in IPsec. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 12–29. Springer Berlin Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_2
44. Rescorla, E., Modadugu, N.: Datagram Transport Layer Security Version 1.2. RFC 6347 (Jan 2012). https://doi.org/10.17487/RFC6347, https://www.rfc-editor.org/info/rfc6347
45. Rescorla, E., Tschofenig, H., Modadugu, N.: The Datagram Transport Layer Security (DTLS) Protocol Version 1.3. RFC 9147 (Apr 2022). https://doi.org/10.17487/RFC9147, https://www.rfc-editor.org/info/rfc9147
46. Rizzo, J., Duong, T.: Beast: Surprising crypto attack against https. In: EKOPARTY security conference 7th edition (2011)
47. Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) ACM CCS 2002. pp. 98–107. ACM Press, Washington, DC, USA (Nov 18–22, 2002). https://doi.org/10.1145/586110.586125
48. Rogaway, P., Zhang, Y.: Simplifying game-based definitions - indistinguishability up to correctness and its application to stateful AE. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 3–32. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0_1
49. Shrimpton, T.: A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272 (2004), https://eprint.iacr.org/2004/272
50. Velvindron, L.: Deprecating RC4 in Secure Shell (SSH). RFC 8758 (Apr 2020). https://doi.org/10.17487/RFC8758, https://www.rfc-editor.org/info/rfc8758

A Ethical Considerations

A.1 Stakeholder Analysis

The results of this work are immediately relevant to protocol designers and standards bodies, such as the IETF, who can use them to improve their understanding and design of internet standards that rely on AEAD and stateful encryption. In addition, they are relevant to security researchers, who may apply comparable methodologies to other protocols. Our findings can help SSH library implementers prioritize the selection of stateful encryption modes, enhance their documentation, and warn users about possible dangers. Our results may further aid in finding appropriate mitigations for any weaknesses similar to the Terrapin attack that may be found in the future. Indirect benefits are also obtained by end users of products that use these protocols.

We acknowledge that adversaries can also benefit from improved understanding of protocol weaknesses; for example, they might be able to identify new attacks more efficiently by misusing the information in this work. In our assessment, the defensive benefits significantly outweigh any potential advantage given to adversaries.

A.2 Impact

In light of ethical standards, we considered how this work would benefit the aforementioned stakeholders, particularly in terms of enhancing protocol security and addressing state-based vulnerabilities in our adversarial model. We also considered the possible advantages of informing the design of future protocols and creating strong defenses against the Terrapin attack and other vulnerabilities of a similar nature. Our study did not involve human subjects, personal information, live system interaction, or other consent- or privacy-related concerns.

Our findings apply to a wide range of protocols and do not disproportionately impact any specific user group. The decision to focus on SSH rather than similar protocols such as TLS, IPsec, DTLS, SRTP, and QUIC reflects the Terrapin attack's focus on SSH. We believe that a thorough examination of a single protocol benefits stakeholders more than a broader but more superficial study.

This work adheres to the established norms of responsible security research and does not present any new practical attacks beyond those in the original Terrapin publication [5], out of respect for law and the public interest. Therefore, we believe that conducting and publishing this research is unlikely to cause direct harm to stakeholders. No responsible disclosure was required, given that the Terrapin attack had already been disclosed in [5].

A.3 Mitigations

Our analysis is abstract and model-based and does not directly lead to exploitable attacks in real implementations. The focus of this work is on classification and understanding of stateful encryption and AEAD modes. These

results can be used to implement robust mitigations against the Terrapin attack and potential future attacks similar to it, as well as systematically harden protocol designs against such vulnerabilities. In this way, our work encourages safer cryptographic design and deployment.

A.4 Decision to Publish

We believe that the stakeholders protecting against attacks would suffer more from withholding our findings than adversaries would. Adversaries currently possess the Terrapin attack method, and their capabilities would not be significantly enhanced by withholding a formal analysis. Defenders, on the other hand, need to systematically find and fix pertinent flaws in protocols and implementations. For these reasons, we conclude that publication of this work is ethically justified.

B Generative AI Usage

For this work, we used generative AI, namely ChatGPT, Claude, and Google Gemini, to proofread the paper and identify possible gaps. All issues flagged by the AI were manually verified and, where necessary, corrected. Furthermore, we used these tools alongside LanguageTool to improve grammar and spelling, and for light style polishing, that is, by rewriting individual sentences for the final submission.

C Formal Model for Indistinguishability of Ciphertexts

Figure 11 is an extension of our Terrapin security model. It replaces the CPA oracle $\mathsf{SND}(m)$ with a left-or-right oracle $\mathsf{SND}(m_0, m_1)$. Note that this security model is strictly stronger than the Terrapin model, since we can use the left-right oracle as a CPA oracle by calling $\mathsf{SND}(m, m)$.

$\mathsf{Exp}^{\text{CIND-PCSA-CCA2}}_{\mathcal{A}}(1^\lambda)$

1 : $(\mathsf{k}, \mathsf{st}_{s,0}) \leftarrow \mathsf{SF.Init}(1^\lambda)$
2 : $\mathsf{st}_{s,1}, \mathsf{st}_r \leftarrow \mathsf{st}_{s,0}$
3 : $b \leftarrow\!\!\$\ \{0,1\}$
4 : $M \leftarrow []$
5 : $j_s, j_r \leftarrow 0$
6 : $p_s, p_r \leftarrow 0$
7 : $f_r \leftarrow 0$
8 : $b' \leftarrow \mathcal{A}^{\mathsf{SND}(\cdot,\cdot),\mathsf{RCV}(\cdot,\cdot),\mathsf{SET}(\cdot)}$
9 : **return** $(b' = b)$

$\mathsf{SND}(m_0, m_1)$

1 : $p_s \leftarrow 1$
2 : $(\mathrm{H}_0, c_0, \mathsf{st}_{s,0}) \leftarrow\!\!\$\ \mathsf{SF.Enc}(\mathsf{k}, \mathsf{st}_{s,0}, m_0)$
3 : $(\mathrm{H}_1, c_1, \mathsf{st}_{s,1}) \leftarrow\!\!\$\ \mathsf{SF.Enc}(\mathsf{k}, \mathsf{st}_{s,1}, m_1)$
4 : $res \leftarrow c_b$
5 : **if** $\mathsf{len}(c_0) \neq \mathsf{len}(c_1)$ **then** $res \leftarrow \perp$
6 : **if** $(c_0 = \perp)\,\mathrm{OR}\,(c_1 = \perp)$ **then** $res \leftarrow \perp$
7 : **if** $res \neq \perp$ **then**
8 : $\quad M[j_s] \leftarrow m_b$
9 : $\quad j_s \leftarrow j_{s+1}$
10 : $\quad$ **return** (H_b, c_b)

$\mathsf{RCV}(\mathrm{H}, c)$

1 : $p_r \leftarrow 1$
2 : **if** $f_r = 1$ **then return** $\perp$
3 : $(m, \mathsf{st}_r) \leftarrow \mathsf{SF.Dec}(\mathsf{k}, \mathsf{st}_r, \mathrm{H}, c)$
4 : **if** $m = \perp$ **then** $f_r \leftarrow 1$; **return** $\perp$
5 : **if** $m \neq M[j_r]$ **then**
6 : $\quad res \leftarrow m$
7 : **else**
8 : $\quad res \leftarrow \perp$
9 : $j_r \leftarrow j_r + 1$
10 : **if** $b = 1$ **then return** res **else return** $\perp$

Fig. 11. Our model for ciphertext indistinguishability under adaptive chosen-ciphertext attacks. The pseudocode defines a single-instance security game played between an adversary $\mathcal{A}$ and a CIND-PCSA-CCA2 challenger, where the adversary is provided access to the SND and RCV oracles as well as the SET oracles from Fig. 2. The array M records which message is allowed at which index in the channel.