Terrapin Attack: Breaking SSH Channel Integrity By Sequence Number Manipulation

Fabian Bäumer
Ruhr University Bochum

Marcus Brinkmann
Ruhr University Bochum

Jörg Schwenk
Ruhr University Bochum

Abstract

The SSH protocol provides secure access to network services, particularly remote terminal login and file transfer within organizational networks and to over 15 million servers on the open internet. SSH uses an authenticated key exchange to establish a *secure channel* between a client and a server, which protects the confidentiality and integrity of messages sent in either direction. The secure channel prevents message manipulation, replay, insertion, deletion, and reordering. At the network level, SSH uses the *Binary Packet Protocol* over TCP.

In this paper, we show that as new encryption algorithms and mitigations were added to SSH, the SSH Binary Packet Protocol is no longer a secure channel: SSH channel integrity (INT-PST, aINT-PTXT, and INT-sfCTF) is broken for three widely used encryption modes. This allows *prefix truncation attacks* where encrypted packets at the beginning of the SSH channel can be deleted without the client or server noticing it. We demonstrate several real-world applications of this attack. We show that we can fully break SSH extension negotiation (RFC 8308), such that an attacker can downgrade the public key algorithms for user authentication or turn off a new countermeasure against keystroke timing attacks introduced in OpenSSH 9.5. Further, we identify an implementation flaw in AsyncSSH that, together with prefix truncation, allows an attacker to redirect the victim's login into a shell controlled by the attacker.

We also performed an internet-wide scan for affected encryption modes and support for extension negotiation. We find that 71.6% of SSH servers support a vulnerable encryption mode, while 63.2% even list it as their preferred choice.

We identify two root causes that enable these attacks: First, the SSH handshake supports optional messages that are not authenticated. Second, SSH does not reset message sequence numbers when activating encryption keys. Based on this analysis, we propose effective and backward-compatible changes to SSH that mitigate our attacks.

1 Introduction

Secure Shell (SSH). While TLS is commonly used to secure user-facing protocols such as web, email, or FTP, SSH is used by administrators to deploy and maintain these servers, often with high privilege (root) access and a large attack surface for lateral movement within an organization's infrastructure. SSH was developed by Tatu Ylonen in 1995 as a secure alternative to telnet and rlogin/rcp and has since become a critical component of internet security.

In 1996, SSHv2 was developed to fix severe vulnerabilities in the original version. In February 1997, the IETF formed the SECSH working group to standardize SSHv2. After a decade, it published five core RFCs [29–33]. SSHv2 provides cryptographic agility and protocol agility without breaking backward compatibility. Since its original release, dozens of standardized and informal updates to the protocol have been published. Because of this, SSHv2 remains relevant after 25 years without major redesign, but it has also become difficult to analyze. There is a significant risk that these extensions of SSH interact to undermine its security goals.

SSH Connections. An SSH connection between a client and a server begins with the *Transport Layer Protocol* [33], which defines the handshake messages for key exchange and server authentication and how messages are exchanged over TCP using the *Binary Packet Protocol* (BPP). After the handshake, SSH provides a *secure channel*[1] for application data. At the application level, the client chooses a sequence of *services* to run. In practice, the client will run precisely two services: the *Authentication Protocol* [30] for user authentication with a password or public key, followed by the *Connection Protocol* [31] for the bulk of SSH's features like terminal sessions, port forwarding, and file transfer.

[1] We note that by *channel*, we refer to the integrity-protected, encrypted byte stream at the transport level, not the SSH data channels from the connection protocol. Multiple SSH data channels can be multiplexed over the same secure transport channel.

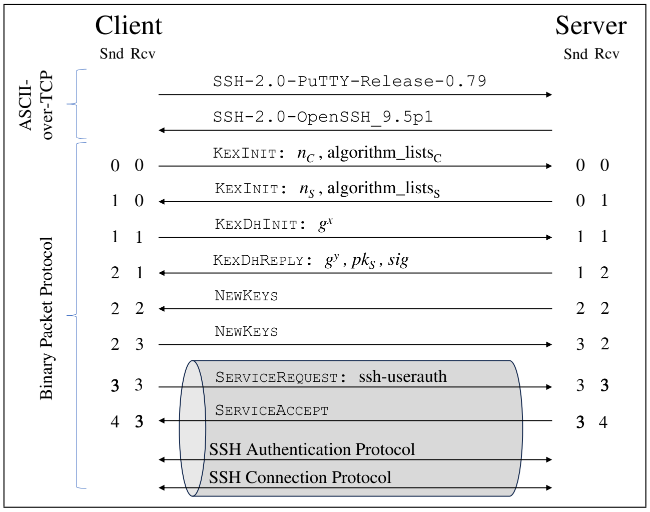


Figure 1: Typical SSH handshake using a finite-field Diffie-Hellman key exchange. Included sequence numbers are implicit and maintained by the BPP. Snd denotes the counter for sent packets and Rcv for received packets. Sequence numbers verified using authenticated encryption are in **bold**.

1.1 SSH Channel Security

In this work, we focus on the integrity of the SSH handshake and the resulting secure channel, as shown in Figure 1. After an initial exchange of version information directly over TCP, the BPP exchanges *packets*, each containing precisely one *message*. Initially, the BPP is used without encryption or authentication for the duration of the handshake until the NewKeys message. Afterward, the encryption and authentication keys are used to form a secure channel, intending to protect the confidentiality and integrity of the *ordered stream of all following messages*. Note that technically, the secure channel consists of two separate cipher streams, one for each direction, and that the order of message arrival is only guaranteed for each direction separately.

Message Authentication Codes. As SSH is an interactive protocol, the integrity of each packet must be verified when it is received so that it can be promptly processed. For this, the BPP appends a Message Authentication Code (MAC) to each packet. A cipher mode and a MAC form an authenticated encryption scheme [5]. SSH historically uses *Encrypt-and-MAC* (EaM), where the MAC is computed over the plaintext, but this is vulnerable to oracle attacks [2]. Later, *Encrypt-then-MAC* (EtM) was added, where the MAC is computed over the ciphertext instead. SSH has recently adopted the AEAD ciphers AES-GCM and ChaCha20-Poly1305, where ciphertext integrity is built into the encryption scheme [43].

A Trivial Example: Suffix Truncation Attacks. Note that a per-packet MAC cannot fully protect the channel's integrity, as packets are verified and decrypted before the end of communication has been seen. This allows for a trivial *suffix truncation attack*, where the attacker interrupts the message flow at some point during the communication. This is an inherent limitation of interactive protocols and an accepted trade-off in the design of SSH, but also, e.g., the TLS Record Layer. Although this attack cannot be prevented, it can be detected by requiring "end-of-communication" messages as the last messages in both directions. TLS defines a "close_notify" alert for this purpose [42]. Although SSH also defines a Disconnect message to indicate the end of the secure channel, this message is optional, unidirectional, and not described as security-critical in the standard.

Implicit Sequence Numbers. If the MAC was only computed over the payload of each packet, an attacker could still delete, replay, or reorder packets. Therefore, a *sequence number* is included in the MAC computation, corresponding to the position of the message in the stream. Each peer maintains two counters (starting at 0), one for each direction. The Snd counter is incremented after a packet has been sent, and the Rcv counter is incremented after a received packet has been processed. Once the secure channel has been established, the current value of Snd is used to compute the MAC of an outgoing packet, and the current value of Rcv is used to verify the MAC of an incoming packet. If packets in the secure channel are deleted, replayed, or reordered, the sequence numbers get out of sync, and MAC verification will fail.

Because TCP is a reliable transport, accidental reordering of SSH packets cannot occur on the network. Thus, SSH (like other TCP-based protocols) uses *implicit sequence numbers* that are not transmitted as part of the packet.

Security Guarantees of Secure Channels. For TLS, the security guarantees of the Record Layer were formalized as stateful length-hiding encryption [39], with the state mainly consisting of the implicit sequence number. The security of the BPP and implicit sequence numbers was analyzed by Bellare et al. in [4] and later refined and extended by Paterson and Watson [40] and Albrecht et al. [1]. These works define, in slightly idealized scenarios, the following informal security goal for a secure channel:

> When a secure channel between A and B is used, the data (or message) stream received by B should be identical to the one sent by A and vice versa (INT-PST, aINT-PTXT in [17]).

Within their idealizations, all three works confirm that the BPP is indeed a secure channel. The difference between the models is that Paterson and Watson [40] also included the encrypted length field of the Encrypt-and-MAC modes, while Albrecht et al. [1] considered the more recent cipher modes ChaCha20-Poly1305, AES-GCM, and generic Encrypt-then-MAC. Our attacks show that the models underlying the proofs in [1] are only partially accurate. We will explain the discrepancies between the proofs and our findings in Section 2.

1.2 Overview of Our Attacks on SSH

In this paper, we show that SSH fails to protect the integrity of the encrypted message stream against meddler-in-the-middle (MitM) attacks. More precisely, we present novel *prefix truncation attacks* against SSH:

> We show that the SSH Binary Packet Protocol is *not* a secure channel because a MitM attacker can delete a chosen number of integrity-protected packets from the beginning of the channel in either or both directions without being detected (Figure 2).

Attacker Model. We consider a MitM attacker who can observe, change, delete, or insert bytes at the TCP layer. We do *not* assume that the attacker can break the confidentiality of the session keys, i.e., the attacker has no information about the derived encryption keys, MAC keys, or IV. However, we do assume that the attacker can determine the length of the messages to be deleted even if the length field is encrypted. We discuss the practicality of this in Section 4.3. The rogue session attack presented in Section 6.2 further assumes the attacker has an account on the same host as the victim.

As for the connection, we assume that the server is correctly authenticated (i.e., the client recognizes the server's host key) and that a vulnerable encryption mode has been negotiated. See Table 1 for a list of vulnerable encryption modes.

Prefix Truncation Attacks. While our attacks on SSH are novel, the idea of prefix truncation attacks against network protocols by sequence number manipulation is not. To the best of our knowledge, the first and only description of such an attack is by Fournet (on behalf of miTLS) in an email to the TLS working group in 2015, targeting a draft version of TLS 1.3 [18]. Fournet's attack increases sequence numbers in TLS by message fragmentation rather than message injection and remains theoretical, as *"prefix truncations will probably cause the handshake to fail."* Subsequently, the draft was modified, and no prefix truncation attacks against the final version of TLS 1.3 are known. In contrast, we present the first real-world, practical prefix truncation attack against a mature, widely used protocol.

Root Cause Analysis. Our results depend on two technical observations about how SSH protects the integrity of the handshake and channel:

1. *SSH does not protect the full handshake transcript.* Although server authentication uses a signature to verify the integrity of the handshake, the signature is formed over a fixed list of handshake messages rather than the complete transcript. This gap in authentication allows an attacker to insert messages into the handshake and thereby manipulate sequence numbers.

2. *SSH does not reset sequence numbers at the beginning of the secure channel.* Instead, SSH increases sequence numbers monotonically, independent of the encryption state. Any manipulation of sequence numbers before the secure channel carries over into the channel.

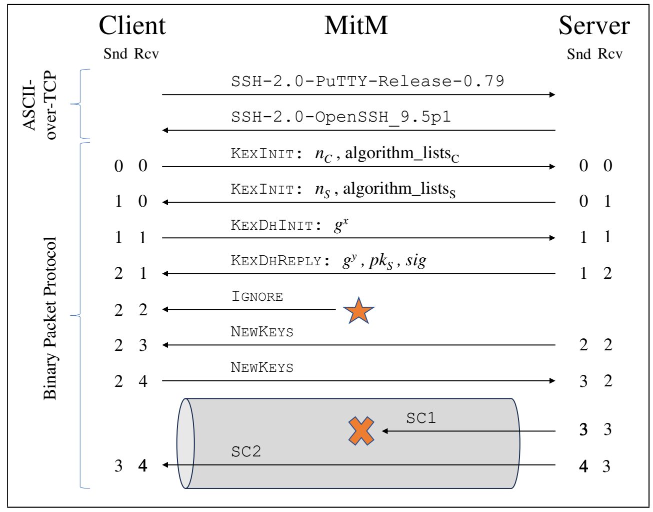


Figure 2: A novel prefix truncation attack on the BPP. The server sends SC1 and SC2, but the client only receives SC2.

Based on these two key observations, we present a series of novel attacks on SSH that increase in complexity and impact.

Sequence Number Manipulation. We show that an attacker can *increase the receive counters* of the server and the client by inserting messages into the handshake. Although not required for any of our attacks, we also show that, for some implementations, an attacker can *fully control the receive and send counters*, setting them to arbitrary values (Section 4.1).

A Prefix Truncation Attack on the BPP. An attacker can use sequence number manipulation to *delete a chosen number of packets at the beginning of the secure channel*. Neither the client nor the server detects this prefix truncation, consequently breaking the channel integrity of SSH (Section 4.2).

Extension Negotiation Downgrade Attack. As a practical example, we show an attack that uses prefix truncation to *break extension negotiation* [9], thereby downgrading the security of the connection. The attacked client might mistakenly believe that the server does not support recent signature algorithms for user authentication or does not implement certain countermeasures to attacks (Section 5.2). Specifically, the attacker can turn off protection against keystroke timing attacks in the recently released OpenSSH 9.5.

Rogue Extension Attack and Rogue Session Attack. As another example, we show two attacks on the AsyncSSH client and server. In the first attack, *the victim's extension info message is replaced* with one chosen by the attacker (Section 6.1). For the second attack, the attacker must have a user account on the same server as the victim. The attacker injects a malicious user authentication message so that *the victim logs into a shell controlled by the attacker* rather than the victim's shell, thereby giving the attacker complete control over the victim's terminal input (Section 6.2). These attacks combine prefix truncation with implementation flaws in the AsyncSSH library.

Limitations. Our attacks critically depend on the SSH encryption mode negotiated between the client and the server. The attack works best with the AEAD cipher ChaCha20-Poly1305 (added in 2013). The attack also works with any EtM mode (added in 2012), although the success probability depends on the cipher mode negotiated. CBC-EtM can be exploited with a significant probability, while the exploitability of CTR-EtM is low. On the other hand, CBC-EaM, CTR-EaM, and GCM modes are not affected. See Section 4.4 for a complete analysis.

In an internet-wide scan, we show that despite these limitations, 71.6% of all SSH servers on the internet support an affected encryption mode, and 63.2% even list it as their preferred choice (Section 7).

1.3 Our Contributions

We contribute the following novel results:

- An analysis of the integrity of SSH channels, where we identify two previously unknown flaws in the SSH specification, namely gaps in the handshake authentication and the use of sequence numbers across key activation.
- A novel prefix truncation attack on SSH channel integrity, where we show that an attacker can manipulate the sequence numbers and delete several messages from the beginning of the secure channel.
- A first security analysis of SSH extension negotiation, including a novel downgrade attack that disables extension negotiation completely. Thus, support for some public key signature algorithms or, with OpenSSH 9.5, protection against keystroke timing attacks can be disabled.
- As a practical demonstration, two novel attacks on AsyncSSH. First, a rogue extension attack, where the attacker can insert a chosen extension negotiation message. Second, a rogue session attack that allows the attacker to log the victim into an attacker-controlled shell. Both escalate implementation flaws in AsyncSSH using the prefix truncation attack.
- An internet-wide scan with up-to-date information on the distribution of SSH encryption modes and extensions.

Artifacts. Proof-of-concept implementations for our attacks and the aggregated results of our internet-wide scan are available under the Apache-2.0 open-source license. See:
`https://github.com/RUB-NDS/Terrapin-Artifacts`

Ethics Consideration and Responsible Disclosure. We disclosed our findings to 33 vendors of SSH implementations, including OpenSSH and AsyncSSH, in October and November 2023, followed by a public disclosure on December 18th, 2023. As of February 2024, 28 vendors have published patches implementing a backward-compatible countermeasure proposed by OpenSSH. The general protocol flaw has been assigned CVE-2023-48795 (CVSSv3 5.9), while the implementation flaws in AsyncSSH were assigned CVE-2023-46445 (Rogue Extension Negotiation; CVSSv3 5.9) and CVE-2023-46446 (Rogue Session Attack; CVSSv3 6.8). To estimate the adoption rate of the countermeasure, we scanned the IPv4 address space on January 5th, 2024, indicating that more than 3.4M servers were patched.

We provide an opt-out option and an email address for inquiries about our internet-wide scans. Additionally, we employ a block list to exclude networks that opted out of previous scans. Scan results are solely published in aggregated form, without any information that could identify individual servers or networks.

2 Related Work

Secure Channels. In 2001, Canetti and Krawczyk [12] established the first model for secure channels, which only requires protection against adversarial insertion of messages. Paterson et al. [39] defined stateful length-hiding authenticated encryption (sLHAE) to model the TLS record layer as a secure channel. This definition was used in [23, 24] to define authenticated and confidential channel establishment (ACCE) to analyze the TLS handshake and record layer as a whole. Bellare et al. [4] used stateful authenticated decryption to define a security notion for SSH that is directed against replay and out-of-order delivery attacks (INT-sfCTXT). Paterson and Watson [40] later refined this work to cover buffered decryption (INT-BSF-CTXT). Albrecht et al. [1] further refined and extended this definition to cover ciphertext fragmentation attacks more generally (INT-sfCTF). Generalizing the work on TLS and SSH, Fishlin et al. [17] defined, among other notions, plaintext integrity for generic data and (atomic) message streams (INT-PST, aINT-PTXT).

Our attacks show that SSH BPP, when instantiated with ChaCha20-Poly1305, CBC-EtM, or CTR-EtM, does not provide integrity of plaintext or ciphertext (message) streams (INT-PST, aINT-PST, INT-sfCTF) as defined in [1, 17].

Truncation Attacks. Suffix truncation attacks against web services using TLS have been demonstrated by Smyth and Pironti in [44]. A prefix truncation attack against a draft version of TLS 1.3 was described by Fournet (on behalf of miTLS) in an email to the TLS working group in 2015 [18]. Fournet's attack increases TLS sequence numbers by message fragmentation rather than injection to avoid breaking handshake authentication. The attack remained theoretical as *"prefix truncations will probably cause the handshake to fail."* As a countermeasure, the draft was changed back to reset sequence numbers to 0 when activating keys.

Attacks on SSH. The most severe attack on SSH was presented by Albrecht, Paterson, and Watson [2] in 2009. It exploited the encrypted length field, using the length of the ciphertext accepted by the server from the network as a decryption oracle for parts of a ciphertext block. In [40], this peculiarity of the BPP was formalized, and in [1] a variant of this attack was presented. Other attacks on SSH include a timing attack on SSH keystrokes by Song, Wagner, and Tian [45], a theoretical attack on SSH CBC cipher modes by Wei Dai [14], and a SHA-1 chosen prefix collision attack on the handshake transcript by Bhargavan and Leurent [8]. The weakness of some SSH host keys presented by Heninger et al. [20] was caused by a lack of entropy and faulty implementations and is not an inherent weakness of the protocol.

Formal Proofs for SSH. The SSH handshake was analyzed by Williams [47] and Bergsma et al. [7]. Bellare et al. [4] presented a generic security model for SSH BPP, and Paterson and Watson [40] a specific, more detailed one for CTR-EaM. Albrecht et al. [1] included security statements for ChaCha20-Poly1305, generic Encrypt-then-MAC, and AES-CTR in SSH, claiming the indistinguishability and integrity of the ciphertext. Careful analysis of their proofs reveals an essential assumption about SSH sequence numbers that does not hold. In particular, they assume that the sequence counters in the stateful encryption scheme are initialized to 0 on both sides, which is false for the cipher modes affected by our attack. This assumption is not apparent from the paper, which omits the pseudocode for the encryption schemes, but Hansen gives the missing parts in [19] (Alg. `ssh-ChaCha20-Poly1305-Gen` in Fig. 6.5 and Alg. `ssh-fgEtM-Gen` in Fig. 6.6 there). We note that this assumption is also present in [4] (Fig. 4 there).

Cadé and Blanchet [11] used the formal verification tool *CryptoVerif* [10] to prove the security of SSH server authentication and the secrecy of the session key in the computational model. The secrecy of messages in the channel cannot be shown due to the attack in [2]. However, they do mention that due to a limitation in the design of CryptoVerif, it cannot keep mutable internal states such as sequence numbers or counters. In their model, the sequence numbers are passed explicitly as arguments and are, therefore, under the attacker's control. The authors do not raise the issue of channel integrity. Other computer-aided proofs of server authentication and secrecy of the session key in the symbolic or computational model can be found in [13, 25], which also do not consider the integrity of the secure channel. For an overview of the field of computer-aided cryptography, see [3].

3 Background

SSH Handshake (Figure 1). To initiate an SSH connection, both peers exchange a version banner. The Binary Packet Protocol (see below) is used from the third message on but without encryption and authentication. In the KexInit messages, nonces and ordered lists of algorithms are exchanged: One list for key exchange, one for server signatures, and two (one per direction) each for encryption, MAC, and compression. For each list, the negotiated algorithm is the first algorithm in the client's list, which is also offered by the server.

In the KexDhInit and KexDhReply messages, a finite-field Diffie-Hellman key exchange is performed. SSH also supports elliptic curves (ECDH) and hybrid schemes with post-quantum cryptography (PQC) as alternatives. The server authenticates itself with a digital signature as part of the handshake. The signature is computed over the contents of the previously exchanged messages in a specified order.

The Exchange Hash: A Partial Handshake Transcript. In contrast to TLS, SSH uses only a selection from the handshake transcript for authentication. The hash value computed from this selection is called *exchange hash* H, defined as

$$H = \text{HASH}(V_C \mid\mid V_S \mid\mid I_C \mid\mid I_S \mid\mid K_S \mid\mid X \mid\mid K),$$

where HASH is the hash function of the negotiated key exchange, V_C and V_S are the version banners of the client and server, I_C and I_S are the KexInit messages, K_S is the server's public host key, and K is the shared secret derived from the key exchange. The value of X depends on the key exchange and contains a composition of negotiated parameters (if any) and the ephemeral public keys of the key exchange [33, Sec. 8]. Each field includes a length field defined by the encoding.

Although the exchange hash contains everything that may influence the negotiation of algorithms or computation of the shared secret, it excludes seemingly 'unimportant' messages or message parts, such as Ignore messages and unrecognized messages. This authentication gap allows a MitM attacker to inject messages into the handshake.

Sequence Numbers. Each sequence number is stored as a 4-byte unsigned integer initialized to zero upon connection. After a binary packet has been sent or received, the corresponding sequence number Snd or Rcv is incremented by one. Sequence numbers are never reset for a connection but roll over to 0 after $2^{32}-1$. As sequence numbers are responsible for protecting against replay attacks, rekeying must occur at least once every 2^{32} packets [37, Sec. 6.1].

We illustrate the use of sequence numbers in Figure 1: After the banner exchange, the counters Snd and Rcv are initialized with $(0,0)$ on both sides. During algorithm negotiation and key exchange, sequence numbers are increased but not used in any MAC computation or verification. Only after keys are activated the secure channel is established, and sequence numbers are used for MAC computation and verification. For each BPP packet, the sequence numbers in bold must match at both peers; otherwise, the BPP packet is rejected.

SSH Binary Packet Protocol. The BPP is used to encrypt and authenticate messages. First, a message is prefixed by a 4-byte message length and a 1-byte padding length. Then, at least 4 bytes of padding are added to the message so that the total length is a multiple of the block size or 8, whatever is larger. On the secure channel, the packet is encrypted by the cipher mode, and a MAC is added. The details depend on the authenticated encryption scheme, which uses an implicit initialization vector IV_{KDF} derived from the session key.

CBC-EaM [33] (Figure 3a) is part of the original SSH specification. The MAC is computed over the implicit sequence number and the packet plaintext. The IV of the first packet is IV_{KDF}, and IV chaining is used (i.e., the IV of packet i is the last ciphertext block of packet $i-1$).

CBC-EtM [36] (Figure 3b) was added to OpenSSH in 2012. Here, the packet length is *not* encrypted to allow checking the MAC before decryption. The MAC is computed over the sequence number, the unencrypted packet length, and the ciphertext. The IVs are handled as with CBC-EaM.

CTR [37] mode was proposed by Bellare, Kohno, and Namprempre [4] as a countermeasure to attacks on CBC with IV chaining. IV_{KDF} is used as the initial counter value and incremented after encrypting a plaintext block. CTR can be used with EaM or EtM, with identical implications for the length field and MAC computation as above.

GCM [22] (Figure 3c) mode was specified by the NSA for Suite B-compliant SSH implementations [21]. Here, ciphertext integrity is part of the cipher mode. The length field is *not* encrypted (solely authenticated) to allow verification of the authentication tag before returning any plaintext. Internally, GCM uses a 12-byte nonce that is initialized to IV_{KDF}. The nonce is split into a 4-byte fixed value and an 8-byte invocation counter that is incremented by one for each message. The sequence number is not used but is always offset by a constant from the invocation counter.

ChaCha20-Poly1305 [34] (Figure 3d) was added to OpenSSH in 2013, inspired by a similar proposal for TLS by Langley and Chang [26, 27]. Here, two different encryption keys for the length field and the packet payload are derived, so the length field cannot be used as a decryption oracle for the payload. The MAC is computed over the concatenation of the two ciphertexts. Internally, the AEAD construction uses the sequence number as a nonce for each packet.

We note that the SSH specification says that the length field is encrypted [33, Sec. 6] and that the sequence number is used for integrity checks [33, Sec. 6.4]. This is only true for CBC-EaM, CTR-EatM, and ChaCha20-Poly1305. The modes CBC-EtM, CTR-EtM, and GCM do not encrypt the length field, and GCM also does not use the sequence number.

4 Breaking SSH Channel Integrity

In this section, we present a novel *prefix truncation attack* on SSH. The basic idea is that the attacker injects messages into the handshake to increase the implicit sequence number in one of the peers and then deletes a corresponding number of messages to that peer at the beginning of the secure channel. Two key insights about the SSH protocol enable this attack:

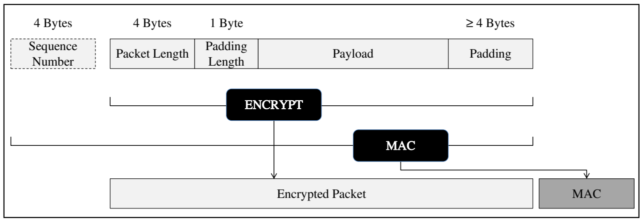
SSH Does Not Protect the Full Handshake Transcript. As detailed in Section 3, the exchange hash signed by the server during the handshake only authenticates some parts of the handshake transcript, while other parts are left unauthenticated. This allows an attacker to inject messages into the handshake, which cannot affect the key exchange but does affect the implicit sequence numbers of the peers.

SSH Does Not Reset Sequence Numbers at the Beginning of the Secure Channel. In SSH, sequence numbers are only incremented and never reset to 0, even when the encryption key changes. This allows an attacker to manipulate the sequence number counters in the secure channel before encryption and authentication keys are activated.

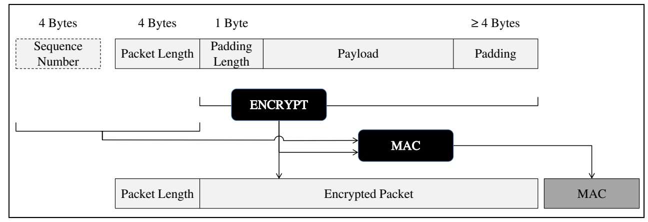
Comparison to Other Protocols. In IPsec/IKE, only a portion of the handshake transcript is signed, but unlike SSH, sequence numbers are reset to 0 when encryption and MAC keys are activated. In TLS, FINISHED messages are exchanged at the beginning of the secure channel to verify the integrity of the complete handshake transcript, and sequence numbers are reset to 0 after installing new keys. The Noise Protocol Framework fully secures the handshake transcript and uses a nonce as a sequence counter that is initialized to 0 after the handshake.

4.1 Sequence Number Manipulation

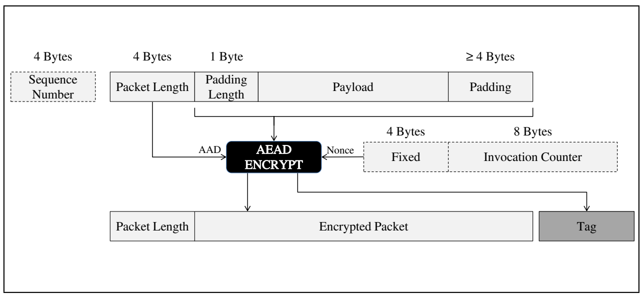
In this section, we first show how a MitM attacker can arbitrarily increase the receive sequence numbers C.Rcv and



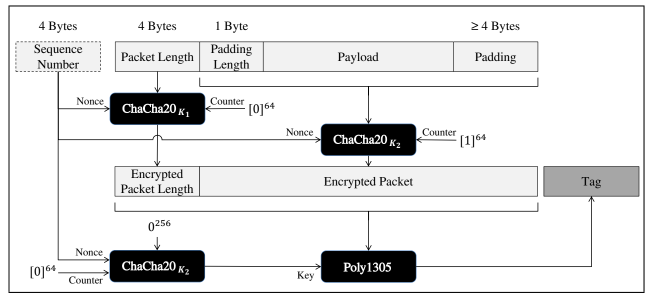
(a) Encrypt-and-MAC [33] (diagram based on [2, Fig. 1]). The packet length is encrypted as part of the packet's plaintext. The sequence number is part of the input used to compute the MAC.



(b) Encrypt-then-MAC [36, Sec. 1.5] (diagram based on [41, Slide 37]). The packet length is not encrypted but authenticated as part of the MAC's input. The sequence number is used similarly to Encrypt-and-MAC.



(c) Galois/Counter Mode [22]. The packet length is not encrypted but authenticated as additional authenticated data (AAD). The sequence number is not used directly but is replaced with an invocation counter of constant offset.



(d) ChaCha20-Poly1305 [34] (diagram based on [41, Slide 39]). The packet length is encrypted using a different key and authenticated as part of the Poly1305 input. The sequence number is encoded as an unsigned 64-bit integer to match the nonce length of ChaCha20.

Figure 3: Commonly used authenticated encryption schemes in the BPP of SSH.

S.Rcv in the client and the server during the handshake. This will be the basis for our prefix truncation attack and its applications, allowing the attacker to compensate for messages deleted from the secure channel.

Technique RcvIncrease (Figure 4a). A MitM attacker can increase C.Rcv (resp. S.Rcv) by N while not changing any other sequence number by sending N IGNORE messages to the client (resp. server).

The correctness is evident from the fact that the SSH standard requires for IGNORE that *"All implementations MUST understand (and ignore) this message at any time."* [33, Sec. 11.2]. The intended purpose of this message is to protect against traffic analysis, so it is considered a security feature, although there is no benefit from it during the handshake phase. We note that the attacker may also use any other message type that does not generate a response.

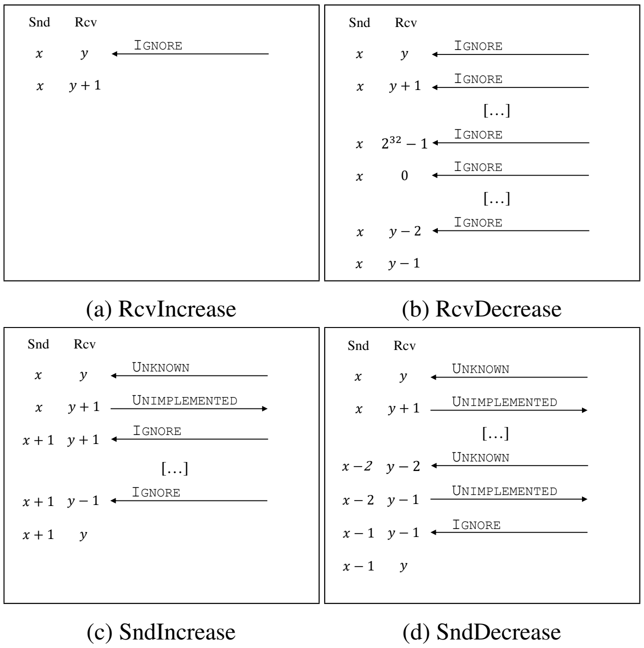
Other Modifications of Sequence Numbers. In addition, we found that an attacker can set the sequence numbers to arbitrary values by using the rollover after 2^{32} messages during the handshake. These advanced techniques require that the implementation allows handshakes with many messages, a large amount of data, and a long operating time. We also require a message that generates a response message but is otherwise ignored. Conveniently, the SSH standard requires this for all messages with unrecognized message IDs [38, Sec. 11.4]. Let UNKNOWN be a message with an unrecognized message ID.

Technique RcvDecrease (Figure 4b). A MitM attacker can decrease C.Rcv (resp. S.Rcv) by N while not changing any other sequence number by sending $2^{32} - N$ IGNORE messages to the client (resp. server).

A single IGNORE message is only 5 bytes, so it fits into a single block even for a 128-bit block cipher. Sending $2^{32} - N$ such messages transfers $\approx 2^{32} \cdot 16\text{B} \approx 69\text{GB}$ of data. Consequently, this technique can fail on implementations with timeouts or restrictions to the amount of data or the number of messages transferred during the handshake.

Technique SndIncrease (Figure 4c) and SndDecrease (Figure 4d). A MitM can increase C.Snd (resp. S.Snd) by N while not changing any other sequence number by sending N UNKNOWN and $2^{32} - N$ IGNORE messages to the client (resp. server) and deleting all generated UNIMPLEMENTED messages. Conversely, a MitM can decrease C.Snd (resp. S.Snd) by N while not changing any other sequence number by sending $2^{32} - N$ UNKNOWN and N IGNORE messages to the client (resp. server) and deleting all generated UNIMPLEMENTED messages.

Here, the total data transfer required is $\approx$ 69GB for SndIncrease and twice as much ($\approx$ 138GB) for SndDecrease. Again,



(a) RcvIncrease (b) RcvDecrease (c) SndIncrease (d) SndDecrease

Figure 4: Techniques for sequence number manipulation as a MitM in the SSH protocol. All techniques can target either client or server before the initial handshake concludes. The MitM deletes all generated UNIMPLEMENTED messages.

these techniques may fail on implementations that have timeouts or restrict the amount of data or number of messages exchanged during the handshake.

Evaluation. We verified all techniques successfully against PuTTY 0.79. Additionally, our experiments show that OpenSSH 9.5p1 recognizes a rollover of sequence numbers and terminates the connection, thus not affected by any technique but RcvIncrease. AsyncSSH 2.13.2 and libssh 0.10.5 allow for RcvIncrease but terminate the connection due to handshake timeouts before any advanced technique concludes. Dropbear 2022.83 disconnects on UNKNOWN messages instead of responding with UNIMPLEMENTED but allows Rcv to roll over, therefore being affected by RcvIncrease and RcvDecrease only.

4.2 Prefix Truncation Attack on the BPP

Single Message Prefix Truncation Attack. We assume the attacker wants to delete the first message SC1 sent from the server (Figure 2). The attack takes two steps:

1. The attacker uses the RcvIncrease technique to increase C.Rcv by one, e.g., by injecting an IGNORE message to the client before NEWKEYS.
2. The attacker deletes the first message SC1 sent by the server.

We first analyze this attack with regard to handshake authentication and sequence numbers. As the key exchange does not protect the handshake transcript from inserting IGNORE messages (Section 3), handshake authentication is not broken. Before the first step, we have $\mathsf{C.Rcv} = \mathsf{S.Snd}$. After the first step, we have $\mathsf{C.Rcv} = \mathsf{S.Snd} + 1$, but this manipulation is not detected during the handshake. After the second step, we have $\mathsf{C.Rcv} = \mathsf{S.Snd}$, and sequence numbers are back in sync.

It remains to be shown that the attacker can delete the message from the channel, which requires knowledge about the message's length, and that its deletion does not affect the MAC verification and decryption output for the following messages. Both aspects require careful analysis with respect to the used encryption mode, which will be given in Section 4.3 and Section 4.4. Here, we conclude by describing a straightforward generalization of the single message attack.

$(\mathbf{N_S}, \mathbf{N_C})$-Prefix Truncation Attack. In a single attack, the attacker can generally delete an arbitrary number of N_S initial messages sent from the server and N_C initial messages sent from the client. This is straightforward: Instead of inserting one IGNORE message to the client before NEWKEYS, the attacker inserts N_S such messages to the client and N_C to the server. Consequently, instead of deleting the first message from the server, the attacker deletes N_S initial messages from the server and N_C initial messages from the client.

Note that the single message attack above is the specific case of a $(1,0)$-prefix truncation attack.

4.3 Determining the Byte-Length of Messages

To successfully delete packets from the secure channel, the attacker has to know their length. This is inherently true for encryption modes that do not encrypt the packet length field (any EtM mode, GCM). In the case of an encryption mode with an encrypted packet length field (any EaM mode, ChaCha20-Poly1305), the attacker may employ different strategies to determine the packet's length. One such strategy is to utilize knowledge about the plaintext if the length of the first few messages inside the secure channel is either fixed (for example, SERVICEACCEPT) or can be measured within a single connection ahead of time (for example, EXTINFO). This approach was used for all attacks described here. More advanced strategies may exploit TCP segment sizes and timings, as well as the message order of the SSH protocol. For example, an attacker may delay all encrypted traffic by the server until after the client's SERVICEREQUEST message has been processed to determine the length of the EXTINFO message. Here, we assume that the attacker always knows the lengths.

4.4 Analysis of Encryption Modes

In this section, we analyze which encryption modes our attacks affect and if they can be exploited in a real-world sce-

Authenticated Encryption Mode		Specification	Enc. State	Dec. State	Affected	Exploitable	Ref.
Encrypt-and-MAC	CBC	[33]	$(IV, \textbf{Snd})$	$(IV, \textbf{Rcv})$	✗	○	Section 4.4.1
	CTR	[33,37]	$(ctr, \textbf{Snd})$	$(ctr, \textbf{Rcv})$	✗	○	Section 4.4.1
Encrypt-then-MAC	CBC	[33,36]	$(IV, \textbf{Snd})$	$(IV, \textbf{Rcv})$	✓	◐	Section 4.4.3
	CTR	[36,37]	$(ctr, \textbf{Snd})$	$(ctr, \textbf{Rcv})$	✓	◐	Section 4.4.3
GCM		[22]	$ctr_{Invocation}$	$ctr_{Invocation}$	✗	○	Section 4.4.1
ChaCha20-Poly1305		[34]	**Snd**	**Rcv**	✓	●	Section 4.4.2

Table 1: Authenticated encryption modes, corresponding specification documents, and their exposure to prefix truncation in the BPP of SSH. The initial value of state variables printed in bold purple can be chosen by the attacker, cf. Section 4.1. Full control of either state enables perfect prefix truncation (●, ChaCha20-Poly1305). Partial control may lead to limited exploitability, depending on the inner workings of the authenticated encryption mode (◐, Encrypt-then-MAC).

nario (see Table 1). An encryption mode is *affected* if, after prefix truncation, all following packets on the secure channel are decrypted, i.e., an AEAD mode does not generate the distinguished symbol INVALID or a composed mode successfully verifies the MAC. Note that we allow decryption to a different plaintext for probabilistic attacks. To capture this, we define an encryption mode as *exploitable* for an attack if the message stream after decryption is well-formed and supports that attack. If the attack's success probability is less than 1, we say the attack has *limited exploitability*.

4.4.1 Not Affected

GCM. GCM [22] mode does not use the implicit sequence number. Instead, it uses an invocation counter, initialized to $\mathrm{IV_{KDF}}$, and incremented after each message. The authors justify this by stating that the resulting nonce is always a fixed offset from the sequence number. By deviating from the SSH standard, GCM stops our attack, as the attacker cannot manipulate the invocation counter during the handshake.

CBC-EaM and CTR-EaM. CBC uses IV chaining, and CTR uses a key stream. When the attacker deletes any prefix of the ciphertext in either mode, the first ciphertext block received will be decrypted as pseudorandom. Because EaM computes the MAC over the plaintext, MAC verification will fail with a probability close to 1, thwarting our attack.

4.4.2 Affected And Perfectly Exploitable

ChaCha20-Poly1305. ChaCha20-Poly1305 [34] directly uses the sequence number in its internal key stream derivation, which makes it vulnerable to our prefix truncation attack. All messages following the truncated prefix are decrypted to their original plaintext because the integrity check of the AEAD cipher is done over the ciphertext and the sequence number, which the attacker has manipulated to match. Under the assumption that the attacker can correctly guess the packet length, the prefix truncation attack always succeeds.

Note that the fault is not with ChaCha20-Poly1305 as an AEAD encryption scheme but with its integration into the SSH secure channel construction.

4.4.3 Affected With Limited Exploitability

CTR-EtM. With CTR-EtM, the MAC is computed over the unencrypted length, the sequence number, and the ciphertext. So, removing some packets from the beginning of the channel does not cause a MAC failure, and cryptographically, the attack succeeds. However, CTR uses a block counter initialized to $\mathrm{IV_{KDF}}$, which increments after each block. After prefix truncation, the key stream is desynchronized, so *all* following ciphertexts are decrypted as pseudorandom packets. Each corrupted packet has a significant probability of causing a critical failure, eventually stopping our attack.

Remark: Decryption Oracle for CTR-EtM Using Prefix Truncation. For CTR-EtM, prefix truncation of k blocks (which exactly contain one or more messages) provides a very limited *decryption oracle* on the ciphertext $c_1, \dots, c_k$ where $c_i := \mathrm{Enc}(\mathrm{IV_{KDF}} + \mathrm{i}) \oplus p_i, 1 \le i \le k$. After deleting the first k blocks, MAC verification for the following message of length l blocks will succeed because the length, sequence number, and ciphertext are correct. The blocks $c_{k+1}, \dots, c_{k+l}$ will be decrypted as $p'_j := \mathrm{Enc}(\mathrm{IV_{KDF}} + \mathrm{j}) \oplus c_{k+j}, 1 \le j \le l$, and processed as a pseudorandom SSH message SC1'. Due to format oracle side channels in SSH at the BPP layer, e.g., the padding length, but also at the protocol layer, e.g., if a message is ignored or triggers a response, the attacker can get some information about the bits in p'_j. This reveals information about the first l key stream blocks, and thus also about $p_1, \dots, p_l$, potentially leaking confidential information like passwords in user authentication. If processing SC1' does not cause a critical failure, the attack can even continue, revealing more about the following key stream and, thus, plaintext. Exploiting this requires a careful study of format oracles in SSH, which is outside the scope of this work.

CBC-EtM. With CBC-EtM, the MAC is computed from the unencrypted length, the sequence number, and the ciphertext. The IV is not required because IV_{KDF} is implicit, and all other IVs are authenticated before use. Consequently, prefix truncation does not cause a MAC failure, and cryptographically, the attack succeeds. Nevertheless, we need to consider the impact that IV chaining has on the immediately following packet to see if this attack is practically exploitable.

Recall that the decryption of the first block is $p_1 := \text{Dec}(c_1) \oplus \text{IV}_{\text{KDF}}$, and for block i, it is $p_i := \text{Dec}(c_i) \oplus c_{i-1}$. We assume the attacker uses prefix truncation to remove blocks $c_1, \ldots, c_k$. The following block c_{k+1} will now be decrypted as $p'_1 := \text{Dec}(c_{k+1}) \oplus \text{IV}_{\text{KDF}}$. We are interested in how SSH implementations process the resulting pseudorandom block p'_1 as the first block in the decrypted packet. Intuitively, it should result in a corrupted packet that causes a critical failure.[2]

Surprisingly, there is a significant probability that the attack can continue, although it is highly implementation-dependent. For a corrupted packet, there are four possible outcomes:

1. *Critically Corrupt:* If corruption is detected at the BPP or application level, e.g., if a length field exceeds the packet length, the connection should be closed.
2. *Marginally Corrupt:* If the packet happens to be similar enough to the original, e.g., if the corruption is limited to optional fields, it should be processed without error and have the same effect as the original would have had.
3. *Evasively Corrupt:* If the packet is well-formed (i.e., has valid padding length) but has an unrecognized message ID, an UNIMPLEMENTED response must be sent, and the connection continues normally [33, Sec. 11.4].
4. Any other case not covered above, in particular, recognized messages different from the original.

Clearly, the first outcome stops any attack from going forward. However, the second, third, and fourth outcomes may be beneficial for the attacker. We will now present two instructive scenarios for outcomes two and three, and estimate the success probability of an attack relying on that outcome. Later, we will verify these estimates experimentally.

Scenario 1: CBC-EtM Prefix Truncation Of a Single Message, Second Message Has Format Flexibility. In this scenario, the attacker wants to remove the first message, and the second (corrupted) message needs to be functionally preserved but has some format flexibility. For example, the second message might be SERVICEACCEPT (see Section 5.2), which is mandatory to start user authentication. The encrypted part of the packet looks like this, where p is the padding length, m is the message ID, and n is the service name length:

p	m	n	Service Name
0e	06	00 00 00 0c	s s h - u s e r a u
t	h	Random Padding	

The probability that the first block decrypts exactly as shown is only 2^{-128} for a 128-bit block cipher. However, for some clients, the service name string is optional. These clients accept a 1-byte message with $p = 30$ (0x1E) and $m = 6$ as *marginally corrupt*, which has a success probability of 2^{-16}, independent of the block size.

Although SERVICEACCEPT may be a lucky case (for the attacker), there are structural reasons for this result: First, SSH messages are often short and can be smaller than a single block. Second, the padding is random and cannot be verified. Third, some messages have redundant fields that implementations ignore (e.g., the service name above).

We experimentally verified that OpenSSH, Dropbear, PuTTY, and libssh allow empty SERVICEACCEPT messages from the server, enabling this attack. At the same time, AsyncSSH is strict by requiring the correct service name.

Scenario 2: CBC-EtM Prefix Truncation Attack On More Than One Message. In this scenario, we assume the attacker wants to remove the first $N > 1$ messages and preserve all the following messages perfectly. Then, the attacker can use prefix truncation to delete the first $N-1$ messages and take a bet on the N-th message to be *evasively corrupt*.

Let ℓ be the length of the ciphertext of the N-th message, with padding length p, message ID m, and random padding. The attack succeeds regardless of the content of the corrupted packet as long as it is well-formed and unrecognized: A packet is *well-formed* if $4 \leq p \leq \ell - 2$ (accounting for the padding length and message ID). A packet is *unrecognized* if m is a message ID not known by the implementation.

Because the message is well-formed, it is not rejected at the BPP layer. Furthermore, because the message is unrecognized, the peer must respond with UNIMPLEMENTED and otherwise ignore it [33, Sec. 11.4], so our attack succeeds.

The probability that a packet is well-formed depends on ℓ. The padding length is between 4 and 255, and ℓ is a multiple of $\max(8, \text{block size})$, so the number of valid padding length values is $\min(252, \ell - 5)$ out of 2^8.

The probability that a packet is unrecognized depends on the size of the set U of unrecognized message IDs in the implementation. The attack requires at least one unknown message ID. Through source code review, we identified 43 IDs that are in active use, so we estimate up to 213 unknown message IDs out of 2^8.

In total, we estimate a success probability of $\min(252, \ell - 5) \cdot |U| \cdot 2^{-16}$. Assuming a block size of at least 128-bit (i.e.,

[2] Similarly to CTR-EtM, any format oracle side channel for p'_1 reveals a relationship between IV_{KDF} and p_{k+1} via $\text{IV}_{\text{KDF}} \oplus p_{k+1} = c_k \oplus p'_1$, which is a marginal information leak for the (secret) IV given information on p_{k+1}, and vice versa. Again, we do not explore this further here.

$\ell \geq 16$), we estimate that the success probability of this attack is between $11 \cdot 2^{-16} \approx 0.0002$ ($\ell_{\text{min}} = 16, |U_{\text{min}}| = 1$) and $252 \cdot 213 \cdot 2^{-16} \approx 0.8190$ ($\ell_{\text{max}} \geq 252, |U_{\text{max}}| = 213$) for vulnerable implementations. Our experiments show success probabilities from 0.0003–0.8383, in good agreement with our analysis (Section 5.2). Increasing the block size increases the lower bound, while the upper bound stays the same.

5 Breaking SSH Extension Negotiation

While the fact that BPP does not implement a secure channel is troublesome enough, exploiting this vulnerability requires an analysis of the SSH protocol after the handshake, i.e., the SSH authentication protocol.

As our attack achieves prefix truncation, it is natural to ask which SSH messages can occur at the beginning of a secure channel. Historically, the first messages exchanged are SERVICEREQUEST and SERVICEACCEPT. Removing either causes the connection to go stale, as the client will not begin the user authentication. Then, our attack, while cryptographically successful, fails at the application layer.

However, the SSH Extension Negotiation mechanism [9] introduces a new message, EXTINFO, which can occur immediately after NEWKEYS as the first message on the secure channel. Some of the extensions that can be negotiated are security-relevant, providing an attack surface for our prefix truncation attack and raising its impact.

In this section, we will first describe SSH Extension Negotiation and then demonstrate how an attacker can downgrade the security of a connection by removing the EXTINFO message from the secure channel in a prefix truncation attack.

5.1 SSH Extension Negotiation

Even though the original SSH RFCs were designed with extensibility in mind, they do not provide any mechanism to negotiate protocol extensions securely. RFC 8308 [9] closes this gap. The RFC describes a signaling mechanism enabling extension negotiation, the extension negotiation mechanism itself, and a set of initially defined extensions.

Support for extension negotiation is signaled as part of the KEXINIT message. The structure of the message is not altered, and the reserved field is not used to avoid compatibility issues. Instead, each peer may include an indicator name within the list of key exchange algorithms. The indicator name differs depending on the role of the peer (`ext-info-c` vs. `ext-info-s`) to avoid accidental negotiation.

Whenever a peer signals support for extension negotiation, the other side may send an EXTINFO message as the first message after NEWKEYS. Additionally, the server can send a second EXTINFO later to authenticated clients to avoid disclosing extension support to unauthenticated clients. Each EXTINFO message can contain several extension entries. Negotiation requirements are defined on a per-extension level.

RFC 8308 defines an initial set of four protocol extensions, and vendors have proposed and implemented additional extensions. We detail those relevant to our attacks here.[3]

`server-sig-algs` [9] is a server-side extension that informs the client about all supported signature algorithms when using a public key during client authentication.

`publickey-hostbound@openssh.com` [35,36] is a server-side extension to advertise support for host-bound public key authentication, which deviates from public key authentication by also covering the server's host key. This allows the enforcement of per-key restrictions when generating the signature outside the SSH client (i.e., when using SSH Agent).

`ping@openssh.com` [36] is a server-side extension to advertise support for a transport-level ping message similar to the Heartbeat extension in TLS [46].

5.2 Extension Downgrade Attack

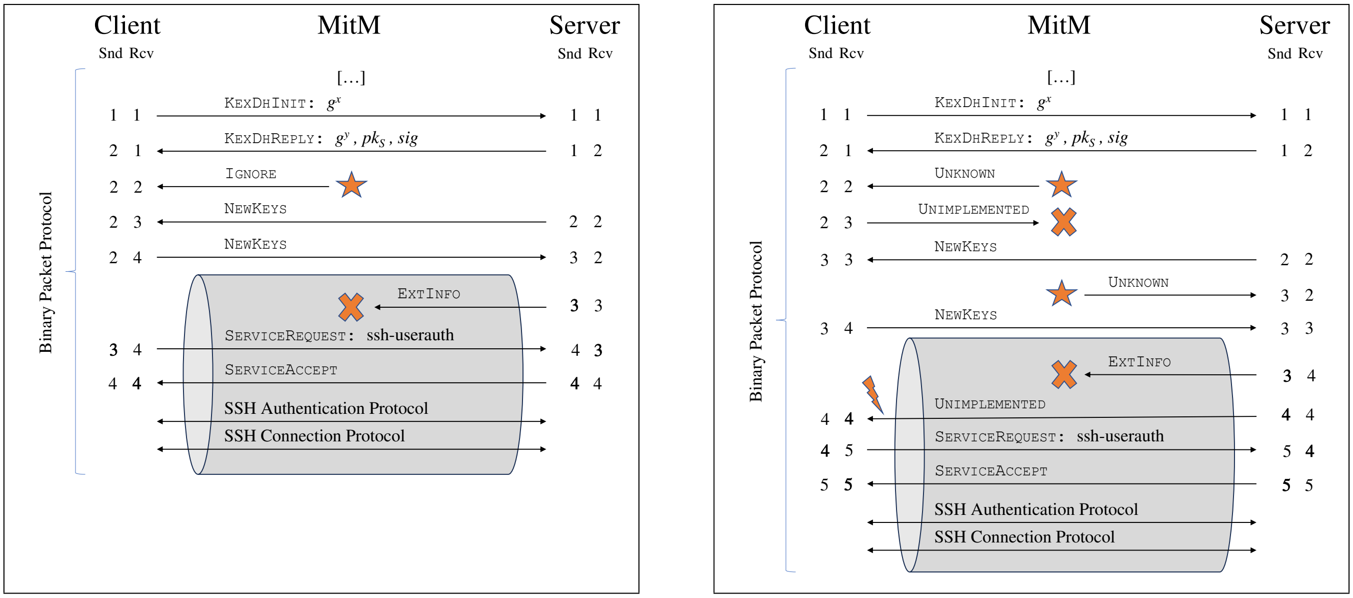
We now show how the prefix truncation attack can be applied to delete the EXTINFO message sent by the client, server, or both parties without either noticing. Our attack differs depending on the encryption mode. For ChaCha20-Poly1305, we can use the basic attack strategy. For CBC-EtM, we show two strategies to generate additional messages in the secure channel so that the attacker can use the "evasively corrupt" outcome of Scenario 2 in Section 4.4.3.

Impact. Successfully performing the extension downgrade can directly impact the security level of the connection. Most notably, the recently introduced keystroke timing countermeasures by OpenSSH 9.5 will remain disabled when the server has not sent `ping@openssh.com`. Furthermore, stripping an EXTINFO containing the `server-sig-algs` extension can lead to a signature downgrade during client authentication, as the client has to resort to trial-and-error instead.

Extension Downgrade for ChaCha20-Poly1305. The downgrade attack for ChaCha20-Poly1305 against the client is depicted in Figure 5a. It is identical to the single message prefix truncation attack from Section 4.2, with EXTINFO now taking the place of SC1 in Figure 2. If the attack should be directed against the server instead, a $(0,1)$-prefix truncation attack should be performed. This allows an attacker to delete any EXTINFO sent immediately after NEWKEYS.

While the server may send a second EXTINFO just before signaling successful client authentication, stripping the EXTINFO message sent after NEWKEYS renders most publicly specified extensions unusable. This is because they are either scoped to the authentication protocol, sent by the client only, or must be sent by both parties to take effect. Solely

[3] We excluded the following extensions because we consider them unrelated to our attacks: `no-flow-control`, `delay-compression`, `elevation`, `global-requests-ok`, `ext-auth-info`



(a) Extension Downgrade Attack for ChaCha20-Poly1305: The MitM injects an IGNORE message before the handshake concludes. The change in sequence numbers allows the MitM to strip the EXTINFO from within the secure channel.

(b) Extension Downgrade Attack for CBC-EtM: The MitM injects UNKNOWN before the NEWKEYS is sent by the client. As the server already sent NEWKEYS, the provoked UNIMPLEMENTED message will be sent within the secure channel after EXTINFO. The corrupted UNIMPLEMENTED message has a significant probability of being ignored (see Scenario 2 in Section 4.4.3).

Figure 5: Variants of the extension downgrade attack for ChaCha20-Poly1305 and CBC-EtM.

the `ping@openssh.com` extension may be sent in the second EXTINFO to enable keystroke timing countermeasures inside the connection protocol. However, OpenSSH 9.5 does not implement any facility to send a second extension negotiation message. As shown in Section 7, extensions scoped to the authentication protocol are the most common among SSH servers on the internet by a significant margin.

Extension Downgrade for CBC-EtM. In Figure 5b, we show how the attack can also work with CBC-EtM. Suppose an attacker injects an UNKNOWN message to the server after the server sends NEWKEYS and EXTINFO but before the client's NEWKEYS message (and also injects UNKNOWN to the client to realign sequence numbers). In that case, the server sends the response UNIMPLEMENTED as the second message in the secure channel immediately after the EXTINFO message. The attacker now wants to remove two messages from the channel and can benefit from the "evasively corrupt" in Scenario 2 in Section 4.4.3. The attacker removes EXTINFO from the secure channel, which causes the decryption of the first block of UNIMPLEMENTED to become pseudorandom. Because UNIMPLEMENTED messages are relatively small ($\ell = 16$ for AES), the upper estimate for the success probability is only $11 \cdot 213 \cdot 2^{-16} \approx 0.0358$.

However, the success probability can be increased significantly by exploiting the new ping extension in OpenSSH 9.5. To make use of this, the attacker replaces the UNKNOWN message sent to the server with a PING message containing at least 255 bytes of payload. As per specification, the server will reflect this data in the PONG response. This yields $\ell \geq 264$, maxing out the probability of the packet being well-formed. Consequently, the upper estimate for the success probability is now $252 \cdot 213 \cdot 2^{-16} \approx 0.8190$.

Evaluation. We successfully evaluated the attack in 10,000 trials on ChaCha20-Poly1305 and CBC-EtM against OpenSSH 9.5p1 and PuTTY 0.79 clients, connecting to OpenSSH 9.4p1 (UNKNOWN only) and 9.5p1. For CBC-EtM, our success rate in practice was 0.0003 (OpenSSH) resp. 0.0300 (PuTTY), improved to 0.0074 (OpenSSH) resp. 0.8383 (PuTTY) when sending PING instead of UNKNOWN.

6 Message Injection Attacks on AsyncSSH

Going beyond the SSH specifications, we now demonstrate how prefix truncation attacks can also be used to exploit implementation flaws. Specifically, we target AsyncSSH,[4] an SSH implementation for Python with an estimated 60k daily downloads.[5] We present two attacks that exploit weaknesses

[4] https://github.com/ronf/asyncssh

[5] https://pypistats.org/packages/asyncssh

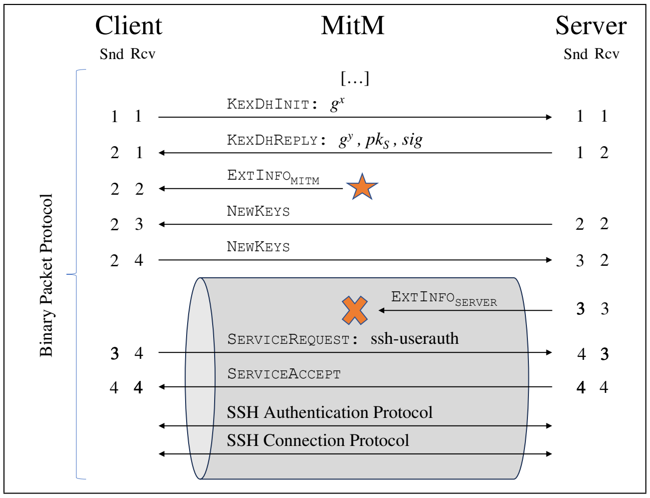


Figure 6: Rogue Extension Negotiation Attack on AsyncSSH: The MitM injects a malicious extension information message before the key exchange completes and deletes the server's EXTINFO message to account for the change in sequence numbers. This attack relates to the generic extension downgrade attack in Section 5.2.

in handling unauthenticated messages during the handshake. These attacks are enabled by prefix truncation and sequence number manipulation.

Note that we describe these attacks only for ChaCha20-Poly1305. Adjusting them for CBC-EtM is straightforward, injecting appropriate IGNORE and UNKNOWN messages, but requires some of the advanced techniques described in Section 4.1. These advanced techniques only work against some SSH implementations.

6.1 Rogue Extension Negotiation Attack

The rogue extension negotiation attack targets an AsyncSSH client connecting to any SSH server sending an EXTINFO message. The attack exploits an implementation flaw in the AsyncSSH client to inject an EXTINFO message chosen by the attacker and a prefix truncation against the server to delete its EXTINFO message, effectively replacing it.

Impact. The attacker can replace the content of the EXTINFO message. AsyncSSH clients support the `server-sig-algs` and `global-requests-ok` extensions. Hence, the attacker can try to downgrade the algorithm used for client authentication by restricting the value of `server-sig-algs` to a subset of those supported by the server.

Attack Description. The attack is a variant of the extension downgrade attack in Section 5.2, but instead of IGNORE, the attacker sends a chosen EXTINFO packet to the client. Similar

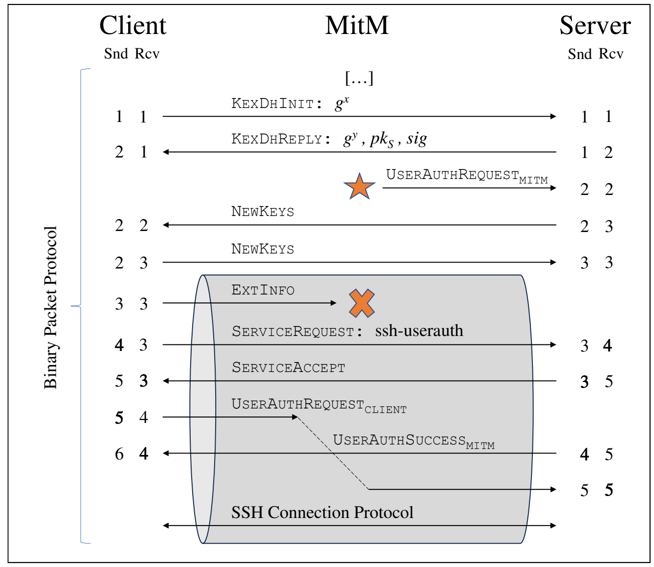


Figure 7: Rogue Session Attack on AsyncSSH: The MitM injects a malicious authentication request before the handshake is complete and deletes the client's EXTINFO message to account for the change in sequence numbers. By delaying the authentication request sent by the client, the MitM ensures that the malicious one is processed. Any additional authentication requests are silently ignored.

to IGNORE, EXTINFO does not trigger a response from the client. A correct SSH implementation should not process an unauthenticated EXTINFO message. However, the injected message is accepted due to flaws in AsyncSSH.

Evaluation. We successfully evaluated the attack against AsyncSSH 2.13.2 as a client, connecting to AsyncSSH 2.13.2.

6.2 Rogue Session Attack

The rogue session attack targets any SSH client connecting to an AsyncSSH server, on which the attacker must have a shell account. The attack's goal is to log the client into the attacker's account without the client being able to detect this.

Impact. With a successful attack, the attacker can gain complete control over the remote end of the SSH session. The attacker receives all keyboard input by the user, completely controls the terminal output of the user's session, can send and receive data to/from forwarded network ports, and can create signatures with a forwarded SSH Agent, if any. The result is a complete break of the confidentiality and integrity of the secure channel, providing a strong vector for a targeted phishing campaign against the user. For example, the attacker can display a password prompt and wait for the user to enter the password, elevating the attacker's position to a MitM at the application layer and enabling impersonation attacks.

Attack Description. The messages exchanged during the attack are depicted in Figure 7. The attacker injects a chosen UserAuthRequest before the client's NewKeys. This request must be a valid authentication request containing the credentials of the attacker. The attacker can use any authentication mechanism that does not require exchanging additional messages between client and server, such as `password` or `publickey`. Due to a state machine flaw, the AsyncSSH server accepts the unauthenticated UserAuthRequest message and defers it until the client has requested the authentication protocol.

To avoid a race condition between the UserAuthRequest sent by the client and the UserAuthRequest injected by the attacker, the attacker delays the client's UserAuthRequest until after the server signals a successful authentication in response to the injected UserAuthRequest. The AsyncSSH server silently ignores any additional authentication request after a successful authentication.

To complete the attack, the attacker has to fix the sequence numbers using one of two strategies (note that Figure 7 only shows the first strategy):

- Suppose the client sends an extra message before ServiceRequest. In that case, the attacker can delete that message from the channel, effectively performing the (0,1)-prefix truncation attack with UserAuthRequest instead of the usual Ignore message.
- Alternatively, suppose the server sends an extra message before ServiceAccept. In that case, the attacker can delete that message after injecting an additional Unknown message to the client before NewKeys, triggering an Unimplemented response that is deleted. This increases both C.Snd and C.Rcv, moving the send count deficit from the client to the server.

Evaluation. We successfully evaluated the attack against AsyncSSH 2.13.2 as a server, connecting to AsyncSSH 2.13.2 and OpenSSH 9.4p1.

7 SSH Deployment Statistics

To estimate the impact of the prefix truncation attacks, we scan for the SSH servers preferring or supporting any affected encryption mode. Similarly, to estimate the impact of the extension downgrade attack, we scan for servers sending ExtInfo messages.

Methodology. For scanning, we used ZMap [16] and ZGrab2 [15] on port 22 of the entire IPv4 address space. The scan was performed over two days in early October 2023, totaling 15.164M SSH servers.

As ZGrab2 cannot capture SSH extensions, we performed a complementary scan at the end of June 2023, using a custom tool, on a subset of 2^{20} open ports. The scan covered a total of 830k servers. All data relating to the use of extension negotiation in SSH is sourced from this scan.

In SSH, the algorithm order of the client determines which algorithm is preferred. However, we cannot scan for actual client use. Assuming that servers and clients are bundled in a single product and share algorithm preference and support, we use the server's lists as a surrogate, as was also done in [1].

Cipher Family	Preferred		Supported	
ChaCha20-Poly1305	8,739k	57.64%	10,247k	67.58%
AES-CTR	4,785k	31.56%	14,866k	98.04%
AES-GCM	1,219k	8.04%	10,450k	68.92%
AES-CBC	236k	1.56%	4,069k	26.84%
Other	147k	0.97%	-	-
Unknown / No KexInit	34k	0.23%	-	-
Total	15,164k	100%		

Table 2: Preferred SSH cipher families as of October 2023.

Symmetric Encryption Algorithms. In Table 2, we show the number of servers that prefer and support various encryption modes. A cipher is preferred if it is placed first in the list of supported algorithms.

We find that, by far, the most preferred encryption cipher is ChaCha20-Poly1305, with 57.64% listing this algorithm first. This is followed by AES-CTR (31.56%) and, with some distance, by AES-GCM (8.04%) and AES-CBC (1.56%).

Authenticated Encryption Modes. As non-AEAD ciphers must be combined with a MAC, we also evaluate which *authenticated* encryption modes the servers prefer and support. The numbers for the AEAD modes ChaCha20-Poly1305 (57.64%) and GCM (8.04%) are identical to those for encryption modes, as the MAC is already integrated. Preference for CTR modes is split between a majority for CTR-EaM (26.14%) and a minority for CTR-EtM (5.46%). Preference for CBC modes is mostly CBC-EaM (2.37%), while preference for CBC-EtM (0.09%) is marginal.

In summary, 63.2% of all servers prefer an authenticated encryption mode affected by our attacks.

Looking at the support for authenticated encryption modes vulnerable to our attacks, we find that 67.58% of all servers support ChaCha20-Poly1305, while 17.24% support CBC-EtM. In total, 71.6% support at least one affected mode.

SSH Extensions. We also looked at SSH extensions offered by servers before user authentication; see Table 4. We can see that 76.81% of all servers send the `server-sig-algs` extensions to indicate support for better signature schemes for client public key authentication. Furthermore, 8.8% send the `publickey-hostbound` extension, improving security

AE Mode	Preferred		Supported	
ChaCha20-Poly1305	8,739k	57.64%	10,247k	67.58%
CTR-EaM	3,964k	26.14%	4,200k	27.70%
GCM	1,219k	8.04%	10,450k	68.92%
CTR-EtM	828k	5.46%	10,685k	70.46%
CBC-EaM	359k	2.37%	1,585k	10.46%
CBC-EtM	14k	0.09%	2,614k	17.24%
Other	2k	0.01%	-	-
Unknown / No KEXINIT	36k	0.24%	-	-
Total	15,164k	100%		

Table 3: Distribution of supported authenticated encryption modes as of October 2023.

Extension name	Times Offered	
`server-sig-algs`	637,466	76.81%
`publickey-hostbound@`	73,040	8.80%
`delay-compression`	283	0.03%
`no-flow-control`	283	0.03%
`global-requests-ok`	283	0.03%

Table 4: SSH extensions offered by servers after the initial handshake, `@openssh.com` abbreviated to `@`. Extensions sent by servers upon successful client authentication are not included.

for authentication using SSH agent. Both extensions provide opportunities for downgrade attacks, as their absence can weaken the strength of the authentication.

8 Suggested Countermeasures

As a stop-gap measure, the affected cipher modes can be turned off. Widely supported alternatives are AES-GCM or AES-CTR. However, the root cause analysis shows that the underlying issues lie in the SSH specification. We therefore suggest two changes to the specification.

Sequence Number Reset. Resetting sequence numbers to zero when encryption keys are activated ensures that sequence number manipulations during the handshake can no longer affect the secure channel. Unfortunately, sequence number reset is a major break in compatibility. To avoid connection failures due to one-sided sequence number resets, we suggest that an implementation signals the support for this countermeasure by including an identification string in the list of supported key exchange algorithms. The SSH extension negotiation mechanism is already employing this method. If and only if both peers signal support for this countermeasure, the sequence numbers will be reset.

In response to our findings, OpenSSH implemented this behavior as part of their so-called "strict kex" countermeasure [36, Sec. 1.10]. In addition to resetting sequence numbers, "strict kex" mandates that unexpected or unknown messages during the initial key exchange must lead to the connection's termination. An unexpected message in this context is any message that is not strictly required for key exchange. "strict kex" has since been adopted by various vendors to ensure interoperability between SSH implementations.

Full Transcript MAC. Authenticating the full handshake transcript, as seen by the client and server, can detect attempts of handshake manipulation by a MitM attacker, including sequence number manipulation through our techniques. It is impossible to extend the scope of the existing exchange hash, as the server signature is transmitted before the new keys are taken into use. Therefore, any messages sent after the key exchange but before NEWKEYS cannot be included. We suggest that both peers send a MAC authenticating the entire transcript at the start of the channel, similar to TLS Finished messages. Signaling support should be done as above. However, the transcript must be carefully canonicalized. While client and server messages are sequential, they can interleave asynchronously, leading to transcript variations. Also, the protocol must be extended to define the algorithm, encoding, and position of the transcript MAC. Thus, securing the handshake is more complex than resetting the sequence number.

Relationship to Formal Proofs. Both countermeasures have a common goal: Align the SSH standard with expectations for stateful encryption schemes from formal models for the BPP presented in [1,4]. A sequence number reset achieves this directly by initializing the sequence numbers to zero, as in the models. On the other hand, verifying the full transcript hash forces the sequence number in the stateful encryption and decryption methods to be synchronized by the sender and receiver. Although the sequence numbers are then not initialized to zero, each pair is nevertheless initialized to a common value out of the attacker's control. The existing models could then be adjusted in the following way: If $T_C, T_S \in \{0,1\}^*$ are the (canonicalized) transcripts of the SSH handshake as seen by the client and the server, and $M_{CS}, M_{SC} : \{0,1\}^* \mapsto \mathbb{N}$ are functions counting the messages from the client to the server and vice versa in a transcript, then sequence numbers in the stateful encryption and decryption modes are initialized to:

$$\mathsf{C.Snd} = M_{CS}(T_C), \qquad \mathsf{C.Rcv} = M_{SC}(T_C),$$
$$\mathsf{S.Snd} = M_{SC}(T_S), \qquad \mathsf{S.Rcv} = M_{CS}(T_S).$$

Authenticating the transcript then ensures that $T_C = T_S$, and thus $\mathsf{C.Snd} = \mathsf{S.Rcv}$ and $\mathsf{C.Rcv} = \mathsf{S.Snd}$, before the first messages in the secure channel are encrypted or decrypted. Authenticating the handshake transcript has the added benefit that the handshake could be analyzed in a "matching conversations"-based security model [23, 24].

Other Issues. We suggest that SSH specifies "end-of-communication" messages to detect suffix truncation attacks. Also, AsyncSSH should be hardened to disallow unauthenticated, application-layer messages during the SSH handshake. In response to our findings, the state machine of AsyncSSH was improved in version 2.14.1 to mitigate our attacks.

9 Future Work

Formally, SSH BPP security was modeled as stateful decryption [1, 4, 40]. Implicitly, this state was associated with SSH sequence numbers, and it was assumed that an adversary could not manipulate this state. These models can be extended in two directions: (1) Include a broader definition of state. By including chained IVs, key stream state, and GCM invocation counters, these models can be used to show why certain cipher modes resist our attacks and that they indeed achieve INT-PST security. (2) Introduce a novel adversarial query, `ModifyState`, to model the attacks described here.

Our attack combines weaknesses in the SSH handshake with weaknesses in the encrypted channel. Earlier work analyzed these separately, leading to small models. To find our attack automatically, models of SSH for computer-aided proofs could (1) model the handshake as well as the BPP together, (2) keep track of sequence numbers in the BPP, including the handshake, which requires modeling integer numbers that can overflow as the internal state, (3) model seemingly unimportant messages like IGNORE, and (4) consider each encryption mode separately. The properties to verify should include strong security notions such as INT-aPTXT [17].

Applying state learning to implementations also has the potential to find our attacks automatically in the future, although it suffers from a combinatorial explosion in the number of messages (see Section 5.1 in [28]). Messages like IGNORE and EXTINFO need to be included in the alphabet to find our attacks, and an active MitM attacker has to be considered.

10 Conclusion

We have shown that the complexity of SSHv2 has increased over its 25 years of development to a point where the addition of new algorithms and features has introduced new vulnerabilities. The root cause analysis has shown that the potential for our attacks was already present in the original specification. Handshake transcripts were never fully authenticated, and sequence numbers were never reset to 0. However, as new authenticated encryption modes and extension messages were added, these weaknesses grew into exploitable vulnerabilities.

We introduced novel sequence number manipulation and prefix truncation attacks for secure channels, which invalidate the INT-aPTXT [17] security of SSH BPP for certain ciphers. We extended these vulnerabilities to real-world exploits like disabling SSH extension negotiation. This yields novel insights into the complex interplay between a practical security mechanism (sequence numbers) and abstract security notions (INT-PTXT vs. INT-CTXT, [6]).

Our close look at the extension negotiation mechanism reveals its design weaknesses: First, sending EXTINFO is optional even if both parties signal support for extension negotiation during the handshake. Second, EXTINFO cannot be used to change the SSH handshake itself, e.g., to implement the countermeasures proposed in this paper. However, it outperforms extension negotiation within the KEXINIT in aspects of privacy as protocol extension can be negotiated securely, i.e., privately, similar to encrypted extensions in TLS 1.3. As a consequence, extension negotiation within the KEXINIT should be strictly limited to extensions affecting the SSH handshake. Protocol extensions affecting the user authentication or application layer should be negotiated through the extension negotiation mechanism.

Although we suggest backward-compatible countermeasures to stop our attacks, the security of the SSH protocol could benefit from a redesign from scratch. The redesign process could be inspired by that of TLS 1.3, which brought implementers together with experts in protocol analysis and formal verification [3]. This could simplify the protocol while preserving and/or achieving desired security notions for SSH, which may differ from those of TLS. For example, while the privacy of client authentication and extension negotiation are relatively new features for TLS, they are already present in SSH and should thus be preserved in a redesign.

Acknowledgements. Fabian Bäumer was supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) project "Industrie 4.0 Recht-Testbed" (13I40V002C). Marcus Brinkmann was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[1] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1480–1491. ACM Press, October 2016.

[2] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26. IEEE Computer Society Press, May 2009.

[3] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. In *2021*

IEEE Symposium on Security and Privacy, pages 777–795. IEEE Computer Society Press, May 2021.

[4] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Authenticated encryption in SSH: Provably fixing the SSH binary packet protocol. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 1–11. ACM Press, November 2002.

[5] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, December 2000.

[6] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008.

[7] Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. Multi-ciphersuite security of the Secure Shell (SSH) protocol. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 369–381. ACM Press, November 2014.

[8] Karthikeyan Bhargavan and Gaëtan Leurent. Transcript collision attacks: Breaking authentication in TLS, IKE and SSH. In *NDSS 2016*. The Internet Society, February 2016.

[9] Denis Bider. Extension Negotiation in the Secure Shell (SSH) Protocol. RFC 8308, March 2018.

[10] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 537–554. Springer, Heidelberg, August 2006.

[11] David Cadé and Bruno Blanchet. From computationally-proved protocol specifications to implementations. In *2012 Seventh International Conference on Availability, Reliability and Security*, pages 65–74, 2012.

[12] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

[13] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. SAPIC+: protocol verifiers of the world, unite! In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3935–3952, Boston, MA, August 2022. USENIX Association.

[14] Wei Dai. email to IETF mailing list. `https://www.ietf.org/ietf-ftp/ietf-mail-archive/secsh/2002-02.mail`, 2002. Accessed: 2023-10-11.

[15] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by internet-wide scanning. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 542–553. ACM Press, October 2015.

[16] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast internet-wide scanning and its security applications. In Samuel T. King, editor, *USENIX Security 2013*, pages 605–620. USENIX Association, August 2013.

[17] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 545–564. Springer, Heidelberg, August 2015.

[18] Cédric Fournet. email to IETF mailing list. `https://mailarchive.ietf.org/arch/msg/tls/extoO9ETJLnEm3MRDTO23x70DFM`, 2015. Accessed: 2023-10-16.

[19] Torben Brandt Hansen. *Cryptographic Security of SSH Encryption Schemes*. Phd thesis, University of London, 2020.

[20] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Tadayoshi Kohno, editor, *USENIX Security 2012*, pages 205–220. USENIX Association, August 2012.

[21] Kevin Igoe. Suite B Cryptographic Suites for Secure Shell (SSH). RFC 6239, May 2011.

[22] Kevin Igoe and Jerome Solinas. AES Galois Counter Mode for the Secure Shell Transport Layer Protocol. RFC 5647, August 2009.

[23] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 273–293. Springer, Heidelberg, August 2012.

[24] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 429–448. Springer, Heidelberg, August 2013.

[25] Pascal Lafourcade and Maxime Puys. Performance evaluations of cryptographic protocols verification tools dealing with algebraic properties. In Joaquin Garcia-Alfaro, Evangelos Kranakis, and Guillaume Bonfante, editors, *Foundations and Practice of Security*, pages 137–155, Cham, 2016. Springer International Publishing.

[26] Adam Langley and Wan-Teh Chang. ChaCha20 and Poly1305 based Cipher Suites for TLS. Internet-Draft draft-agl-tls-chacha20poly1305-04, Internet Engineering Task Force, November 2013. Work in Progress.

[27] Adam Langley, Wan-Teh Chang, Nikos Mavrogiannopoulos, Joachim Strombergson, and Simon Josefsson. ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS). RFC 7905, June 2016.

[28] Eric Lesiuta, Victor Bandur, and Mark Lawford. Slime: State learning in the middle of everything for tool-assisted vulnerability detection. In *Computer Security. ESORICS 2022 International Workshops*, pages 686–704, Cham, 2023. Springer International Publishing.

[29] Chris M. Lonvick and Sami Lehtinen. The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250, January 2006.

[30] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Authentication Protocol. RFC 4252, January 2006.

[31] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Connection Protocol. RFC 4254, January 2006.

[32] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006.

[33] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, January 2006.

[34] Damien Miller. This document describes the chacha20-poly1305@openssh.com authenticated encryption cipher supported by openssh. https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/usr.bin/ssh/PROTOCOL.chacha20poly1305?rev=1.5. Accessed: 2023-10-18.

[35] Damien Miller. SSH agent restriction. https://www.openssh.com/agent-restrict.html, 2022. Accessed: 2023-10-17.

[36] Damien Miller, Markus Friedl, Mike Frysinger, Todd C. Miller, and Darren Tucker. This documents openssh's deviations and extensions to the published ssh protocol. https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/usr.bin/ssh/PROTOCOL?rev=1.55. Accessed: 2024-02-18.

[37] Chanathip Namprempre, Tadayoshi Kohno, and Mihir Bellare. The Secure Shell (SSH) Transport Layer Encryption Modes. RFC 4344, January 2006.

[38] Richard Ogier. OSPF Database Exchange Summary List Optimization. RFC 5243, May 2008.

[39] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 372–389. Springer, Heidelberg, December 2011.

[40] Kenneth G. Paterson and Gaven J. Watson. Plaintext-dependent decryption: A formal security treatment of SSH-CTR. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 345–361. Springer, Heidelberg, May / June 2010.

[41] Kenny Paterson. Advanced security notions for the SSH secure channel: theory and practice. https://summerschool-croatia.cs.ru.nl/2017/slides/Advanced%20security%20notions%20for%20the%20SSH%20secure%20channel.pdf, 2017. Accessed: 2024-02-16.

[42] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[43] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

[44] Ben Smyth and Alfredo Pironti. Truncating TLS connections to violate beliefs in web applications. In *7th USENIX Workshop on Offensive Technologies (WOOT 13)*, Washington, D.C., August 2013. USENIX Association.

[45] Dawn Xiaodong Song, David A. Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In Dan S. Wallach, editor, *USENIX Security 2001*. USENIX Association, August 2001.

[46] Michael Williams, Michael Tüxen, and Robin Seggelmann. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. RFC 6520, February 2012.

[47] Stephen C. Williams. Analysis of the SSH key exchange protocol. In Liqun Chen, editor, *13th IMA International Conference on Cryptography and Coding*, volume 7089 of *LNCS*, pages 356–374. Springer, Heidelberg, December 2011.