Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 14 May 2026

D. Miller
OpenSSH

10 November 2025

SSH Strict KEX extension
draft-ietf-sshm-strict-kex-01

Abstract

This document describes a small set of modifications to the Secure Shell (SSH) protocol to fix the so-called Terrapin Attack on the initial key exchange.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Secure Shell (SSH) is a cryptographic protocol for secure remote connections and login over untrusted networks. The SSH transport layer [RFC4253] uses symmetric encryption to provide a confidential and integrity-protected channel over which application traffic is carried. This transport receives its keys from an initial key agreement sub-protocol, referred to as "key exchange" in the original standards and usually abbreviated as "KEX".

In late 2023, researchers from Ruhr University Bochum identified a novel cryptographic attack [TERRAPIN] on the SSH transport layer and initial key agreement phase. This attack, briefly summarised below, depends on assumptions made by the transport layer and unforeseen interactions between the unencrypted pre-KEX transport and the encrypted post-KEX transport.

In response to this, many SSH implementation deployed the modifications to the SSH transport protocol and KEX sub-protocol described in this document, collectively referred to as "strict KEX". These modifications provide a minimally invasive but comprehensive defence against the Terrapin attack.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Terrapin Attack

2.1. Protocol background

The SSH protocol begins with the client and server exchanging string banners that contain the protocol version and an implementation identifier. Immediately after this, the packetised SSH transport protocol begins. This transport is unencrypted until the initial key exchange (KEX) completes, after which the keys agreed by the endpoints are taken into use and the transport is encrypted.

The SSH transport protocol consists of an ordered sequence of packets. Each SSH transport packet has an explicit type and length, but an implicit sequence number. The sequence number plays a number of roles in the protocol, but is does not appear on the wire and is not explicitly checked. Moreover, the sequence number was not originally specified to ever be explicitly reset on the completion of KEX.

When the transport protocol begins, it is typical for the endpoints to commence KEX immediately by each sending SSH_MSG_KEXINIT packets and only sending packets relevant to the KEX sub-protocol until it concludes with the SSH_MSG_NEWKEYS message. However, sending only KEX-relevant messages during KEX was not strictly required, and a conformant implementation would be expected to accept and process packets such as SSH_MSG_IGNORE or SSH_MSG_DEBUG.

KEX completes when both endpoints perform key derivation, send the SSH_MSG_NEWKEYS message and take the derived keys into use to prepare the symmetric cipher+MAC or AEAD for the transport. This key derivation depends on an "exchange hash" that is made over main values exchanged or derived during the early protocol: banners, SSH_MSG_KEXINIT packet bodies, DH/ECDH/KEM public values and the shared secret derived by the negotiated key agreement method. However, this hash is only taken over selected values and not over every message exchanged between the endpoints (i.e. it is not a "full session transcript").

2.2. Attack Mechanics

The Terrapin attack exploits these preconditions to offer an on-path adversary (a.k.a MITM) two unexpected and undesirable capabilities: 1) the ability to insert messages into the initial key exchange without detection and 2) the ability to selectively delete of one or more consecutive messages from the beginning of the post-KEX transport protocol, despite this supposedly being confidential and integrity-protected by the transport's symmetric cryptography

To perform this attack, an on-path adversary injects one or more packets (such as SSH_MSG_IGNORE) between the SSH banner and initial SSH_MSG_KEXINIT packet, or between other packets before the conclusion of the KEX sub-protocol. These inserted packets should be functionally ignored by the peer but will have the side-effect of incrementing the peer's implicit sequence number.

After the KEX sub-protocol completes, the on-path attacker must then delete an equal number of packets to those that they previously inserted. They may succeed in doing this because the implicit sequence number of the first packet after the deletion will now match the peer's expectation, given the manipulation that occurred by the packets they injected previously. Neither will this manipulation be detected by the KEX exchange hash, as this is over only selected values from the initial protocol phase and not the legal-but-unexpected messages that the attacker inserted.

To demonstrate this attack, first consider the following sequence of packets, which are fairly typical for a SSH server to send to a client during the initial phase of the transport protocol. It lists the sequence numbers as sent by the server and those expected by the client.

```
    sent_seq=0       SSH_MSG_KEXINIT            expected_seq=0
    sent_seq=1       SSH_MSG_KEX_ECDH_REPLY     expected_seq=1
    sent_seq=2       SSH_MSG_NEWKEYS            expected_seq=2
             ---- encrypted transport begins ----
    send_seq=3       SSH2_MSG_EXT_INFO          expected_seq=3
    sent_seq=4       SSH2_MSG_SERVICE_ACCEPT    expected_seq=4
```

Following is an example of a Terrapin attack on this transport:

```
       sent_seq=0       SSH_MSG_KEXINIT           expected_seq=0
       ATTACKER INSERT  SSH_MSG_IGNORE            expected_seq=1
       sent_seq=1       SSH_MSG_KEX_ECDH_REPLY    expected_seq=2
       sent_seq=2       SSH_MSG_NEWKEYS           expected_seq=3
                ---- encrypted transport begins ----
       ATTACKER DELETE  SSH2_MSG_EXT_INFO         [client doesn't see]
       sent_seq=4       SSH2_MSG_SERVICE_ACCEPT   expected_seq=4
```

Note how the attacker is able to desynchronise the client's sequence number by inserting a SSH_MSG_IGNORE message. This insertion is not detectable prior to the commencement of the encrypted transport because sequence numbers are implicit and not checkable. The deletion is not similarly detectable because it resynchronises the sequence number with the client's expectation.

Successfully performing this attack also depends on the symmetric cryptography in use, and it is not practical to achieve for many potential algorithm choices.

Any CBC mode cipher or CTR mode cipher used with the original SSH encrypt-and-MAC construction is immune to this (with cryptographic probability), as the message deletion will desynchronise the ciphertext stream. AES-GCM [RFC5647] is also immune to this as it uses an internal instance counter, that does effectively reset when KEX completes, instead of the SSH transport sequence number.

However this attack is possible for a number of vendor extension algorithms, some very popular across SSH implementations.

The chacha20-poly1305@openssh.com AEAD uses the sequence number as an initialisation vector (IV) to generate its per-packet MAC key and is otherwise stateless between packets. This AEAD is vulnerable as there is no state other than the IV to desynchronise. At the time of publication of the Terrapin attack, this mode was the most popular default cipher for SSH servers ([TERRAPIN] table 2).

The *-etm@openssh.com MAC modes when used with CBC mode ciphers can be exploited with high probability as the desynchronisation in the keystream is limited to the block following the deletion only. CTR mode ciphers used with this MAC mode can not practically be exploited as the keystream is permanently desynchronised after the deletion.

2.3. Attack Impact

As mentioned previously, the Terrapin attack allows 1) undetectable insertion of packets prior to the completion of the initial key exchange and 2) selective deletion of one or more consecutive packets from the initial encrypted SSH transport.

Insertion of additional packets should be harmless, as implementations would be expected to either ignore them or terminate the connection. However, some SSH protocol implementations were found to suffer from serious state machine vulnerabilities that could be attacked by this mechanism ([TERRAPIN] section 6).

Deletion of packets from after the conclusion of the initial key exchange also has a notable impact. Fortunately, most packets from this phase of the protocol are necessary for it to successfully proceed, and so deleting them will cause the connection to fail.

For the client, typically the first messages of the encrypted transport are an optional SSH_MSG_EXT_INFO followed by a SSH_MSG_SERVICE_REQUEST to initiate user authentication. If the SSH_MSG_EXT_INFO was sent by the client, then its deletion by a successful Terrapin attack would not be noticed by the server. However, deleting the SSH_MSG_SERVICE_REQUEST would almost certainly cause the connection to fail, as the user authentication phase that is necessary for all popular SSH implementation would never be initiated.

The server follows a very similar pattern for its early messages over the encrypted transport: an optional SSH_MSG_EXT_INFO followed by a SSH_MSG_SERVICE_ACCEPT reply to the client’s request to start user authentication. Again, the SSH_MSG_EXT_INFO is the only message that could be safely deleted. Most client implementations expect the SSH_MSG_SERVICE_ACCEPT before they will start sending the user authentication requests needed to advance the protocol.

So the Terrapin attack practically allows, subject to implementation and symmetric algorithm choice, the ability to delete a SSH_MSG_EXT_INFO from either the client, server or both.

SSH_MSG_EXT_INFO is defined in [RFC8308] as a mechanism to pass additional information between the client and server that cannot be communicated in the initial SSH key exchange. This information is passed as an array of { key, value } pairs, with several keys defined in Section 3 of [RFC8308]. In addition to these, some SSH implementation use this mechanism to signal support for vendor extensions.

Of the fields defined by [RFC8308], the "server-sig-algs" option is the most relevant to this attack. Deleting this field in a SSH_MSG_EXT_INFO sent by the server could conceivably cause the client to use a weaker signature algorithm during user authentication, though it is difficult to see whether this would have any real-world impact as the signature would still be subject to the confidentiality and integrity protection of the encrypted SSH transport protocol.

An OpenSSH vendor extension, ping@openssh.com is somewhat more interesting as an attack target. This key in a SSH_MSG_EXT_INFO message signals support for a transport-level echo mechanism used by OpenSSH as part of a defence against keystroke timing traffic analysis. Deleting the SSH_MSG_EXT_INFO that signals the presence of this feature would disable this countermeasure.

3. Strict KEX

Strict KEX is a set of two small SSH transport protocol changes to prevent the Terrapin attack: disallowing unnecessary messages prior to the completion of the initial key exchange, and changing the SSH transport protocol to reset the sequence number at the conclusion of the initial KEX and each subsequent KEX.

By disallowing messages that are not strictly required, this extension greatly limits the ability of an on-path adversary to inject data into the SSH transport that is not included in the exchange hash. In the presence of this modification, an attacker can no longer send arbitrary messages to change the sequence number.

Resetting the sequence number after KEX completes eliminates the key piece of implicit transport state that Terrapin depends upon from persisting from the period before the connection is confidential and integrity-protected to after.

3.1. Signaling support for strict KEX

Support for strict KEX is signaled by the presence of new extension marker pseudo-algorithms in the kex_algorithms field of the client and server’s initial SSH_MSG_KEXINIT packet, analogous to how ext-info-c and ext-info-s in this field indicate support for the [RFC8308] SSH_MSG_EXT_INFO in the client and server respectively.

Specifically, a client indicates support for this extension by including either the standard "kex-strict-c" identifier and/or the pre-standard "kex-strict-c-v00@openssh.com" identifier in the kex_algorithms field of the initial SSH_MSG_KEXINIT packet.

Similarly, a server indicates support by including either the standard "kex-strict-s" identifier and/or the pre-standard "kex-strict-s-v00@openssh.com" identifier in its kex_algorithms field.

If the client advertises support for the "kex-strict-c" extension and the server advertises support for the "kex-strict-s" extension, then both endpoints MUST enable the transport protocol changes described below for the duration of the connection.

Similarly, if the client offers support for the pre-standard extension name "kex-strict-c-v00@openssh.com" and the server advertises "kex-strict-s-v00@openssh.com" then both ends MUST enable the protocol changes below.

SSH implementations MUST NOT enable Strict KEX if one offers only the standard name (i.e. "kex-strict-[cs]") and the other offers only the pre-standard name ("kex-strict-[cs]-v00@openssh.com").

Implementations seeking these protections with maximum interoperability SHOULD offer both the standard and pre-standard names, as support for Strict KEX is widely deployed under the pre-standard names.

Finally, the "kex-strict-*" pseudo-algorithm identifiers are valid only in the initial SSH_MSG_KEXINIT message from each endpoint. Their presence or absence in subsequent SSH_MSG_KEXINIT packets MUST be ignored by all parties.

3.2. Disallowing non-KEX messages in initial KEX

When strict KEX is enabled, implementations MUST terminate the connection if they receive a non-KEX message during the initial key exchange. Permitted messages include only SSH_MSG_KEXINIT, SSH_MSG_NEWKEYS and the messages specific to each KEX algorithm:

* SSH_MSG_KEXDH_INIT and SSH_MSG_KEXDH_REPLY for the modp-DH diffie-hellman-* algorithms (Section 8 of [RFC4253]).

* SSH_MSG_KEX_DH_GEX_REQUEST_OLD, SSH_MSG_KEX_DH_GEX_REQUEST, SSH_MSG_KEX_DH_GEX_GROUP, SSH_MSG_KEX_DH_GEX_INIT and SSH_MSG_KEX_DH_GEX_REPLY for the Diffie Hellman group exchange diffie-hellman-group-exchange-* algorithms (Section 5 of [RFC4419]).

* SSH_MSG_KEX_ECDH_INIT and SSH_MSG_KEX_ECDH_REPLY for ECDH KEX algorithms defined in (Section 7.1 of [RFC5656]) and the hybrid Streamlined NTRUPrime/X25519 post-quantum KEM ([I-D.ietf-sshm-ntruprime-ssh]).

* SSH_MSG_KEX_HYBRID_INIT and SSH_MSG_KEX_HYBRID_REPLY for the hybrid ML-KEM/ECDH algorithms ([I-D.ietf-sshm-mlkem-hybrid-kex]).

Because the message that signals support for strict KEX is enabled by the SSH_MSG_KEXINIT message, implementations MUST verify that the SSH_MSG_KEXINIT was the first message received from the peer. Additionally, implementations MUST ensure that the sequence number does not wrap (by incrementing past 2^32-1) at any time prior to the completion of the initial KEX phase. These checks are noted separately, because they must happen somewhat retrospectively, unlike the other enforcement mentioned in this section.

Finally, implementations MUST additionally ensure that any message permitted during KEX can be only accepted the expected number of times. For example, for ECDH KEX, the SSH_MSG_KEX_ECDH_INIT will only be sent a single time by a well-behaved client. A server implementing this extension MUST only accept it once.

3.3. Resetting sequence number at KEX completion

When strict KEX is enabled, both the client and server MUST reset their sequence numbers at the conclusion of the initial KEX and for each subsequent KEX. The sequence point for this reset is after SSH_MSG_NEWKEYS.

Specifically, the sequence number used when sending packets MUST be reset to zero immediately after any SSH_MSG_NEWKEYS packet is sent.

Likewise, the expected sequence number for packets received from the peer MUST be reset after a SSH_MSG_NEWKEYS is received.

One place a sequence number may appear on the wire is the SSH_MSG_UNIMPLEMENTED reply (Section 11.4 of [RFC4253]) to unrecognised messages. There is no special handling of the sequence number in this packet when strict KEX is active - it will use the the same sequence number as the transport packets. I.e. if the first packet sent by an endpoint after SSH_MSG_NEWKEYS was unrecognised, then the sequence number that refers to it in SSH_MSG_UNIMPLEMENTED should be 0.

When strict KEX is enabled, there should be no ambiguity in which packet elicited SSH_MSG_UNIMPLEMENTED. The last paragraphs of Section 7.1 of [RFC4253] require endpoints drain most non-KEX messages before synchronously completing key exchange, and strict KEX requires sequence number reset only on SSH_MSG_NEWKEYS (which cannot be unrecognised), so there is no possibility of an unrecognised message and its reply spanning a sequence number reset.

4. IANA Considerations

This protocol requires one existing registry to be modified.

4.1. Additions to SSH Extension Names

IANA is requested to insert the following entries into the table Key Exchange Method Names [IANA-SSH-EXT] under Secure Shell (SSH) Protocol Parameters [RFC4250].

Method name	Reference	OK to Implement
kex-strict-c	Section 3.1	SHOULD
kex-strict-s	Section 3.1	SHOULD

Table 1

5. Security Considerations

This document describes a number of modifications to the SSH transport protocol to defend against a demonstrated attack that may be performed by active on-path adversaries. While the practical impact of this attack is relatively limited, it does represent a significant violation of the properties expected by a cryptographic protocol and is therefore worth repairing.

These countermeasures are a comprehensive defence to the specific Terrapin attack, but also harden the protocol against other attacks on the initial key agreement phase and the interaction between the pre- and post-KEX transport protocols.

The susceptibility of the original SSH protocol to the Terrapin attack may serve as a demonstration of the danger of retaining implicit state across protocol security boundaries - here, from the unencypted pre-KEX transport to the post-KEX encrypted transport, and also show the desirability of authenticating all messages sent by all parties in the process of key agreement, e.g. using a mechanism like TLS 1.3’s Transcript Hash (Section 4.4.1 of [RFC8446]).

6. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

The following example projects maintain an implementation of this protocol:

OpenSSH OpenSSH is the originating implementation of this extension and has supported it since 2023.

Website: https://www.openssh.com/

PuTTY PuTTY is a popular SSH client implementation for multiple platforms that added strict KEX support in 2023.

Website: https://www.chiark.greenend.org.uk/~sgtatham/putty/

Dropbear Dropbear is a SSH client and server implementation for Unix- like systems. It has supported the strict KEX extension since 2023.

Website: https://matt.ucc.asn.au/dropbear/dropbear.html

Paramiko Paramiko is a SSH client and server implementation in the Python programming language. It has supported the strict KEX modifications since 2023.

Website: https://www.paramiko.org/

Golang x/crypto/ssh The Go programming language project has supported strict KEX in its external "x" repository since 2023.

Website: https://pkg.go.dev/golang.org/x/crypto/ssh

Russh Russsh has implemented strict KEX since 2023.

Website: https://github.com/Eugeny/russh

This list is not exhaustive.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, DOI 10.17487/RFC4250, January 2006, <https://www.rfc-editor.org/info/rfc4250>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <https://www.rfc-editor.org/info/rfc4419>.

[RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, DOI 10.17487/RFC5656, December 2009, <https://www.rfc-editor.org/info/rfc5656>.

[RFC5647] Igoe, K. and J. Solinas, "AES Galois Counter Mode for the Secure Shell Transport Layer Protocol", RFC 5647, DOI 10.17487/RFC5647, August 2009, <https://www.rfc-editor.org/info/rfc5647>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8308] Bider, D., "Extension Negotiation in the Secure Shell (SSH) Protocol", RFC 8308, DOI 10.17487/RFC8308, March 2018, <https://www.rfc-editor.org/info/rfc8308>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[I-D.ietf-sshm-ntruprime-ssh]
Friedl, M., Mojzis, J., and S. Josefsson, "Secure Shell (SSH) Key Exchange Method Using Hybrid Streamlined NTRU Prime sntrup761 and X25519 with SHA-512: sntrup761x25519-sha512", Work in Progress, Internet-Draft, draft-ietf-sshm-ntruprime-ssh-01, 9 December 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-sshm-ntruprime-ssh-01>.

[I-D.ietf-sshm-mlkem-hybrid-kex]
Kampanakis, P., Stebila, D., and T. Hansen, "PQ/T Hybrid Key Exchange in SSH", Work in Progress, Internet-Draft, draft-ietf-sshm-mlkem-hybrid-kex-00, 29 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-sshm-mlkem-hybrid-kex-00>.

7.2. Informative References

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[IANA-SSH-EXT]
IANA, "Key Exchange Method Names", <https://www.iana.org/assignments/ssh-parameters/>.

[TERRAPIN] B辰umer, F., Brinkmann, M., and J. Schwenk, "Terrapin Attack: Breaking SSH Channel Integrity By Sequence Number Manipulation", 2024, <https://arxiv.org/abs/2312.12422>.

Acknowledgments

Thanks Fabian B辰umer, Marcus Brinkmann and J旦rg Schwenk for identifying the Terrapin attack and consulting on the countermeasures described in this document.

These changes were developed with assistance from Darren Tucker, Markus Friedl and Theo de Raadt from the OpenSSH project. Additionally, Simon Tatham of the PuTTY project, Ron Frederick of the AsyncSSH project and Roland Shoemaker of the Golang Security Team offered valuable feedback on the specification of the extension.

Author’s Address

Damien Miller
OpenSSH
Email: djm@openssh.com
URI: https://www.openssh.com/