Internet Engineering Task Force (IETF)
Request for Comments: 8308
Updates: 4251, 4252, 4253, 4254
Category: Standards Track
ISSN: 2070-1721

D. Bider
Bitvise Limited

March 2018

Extension Negotiation in the Secure Shell (SSH) Protocol

Abstract

This memo updates RFCs 4251, 4252, 4253, and 4254 by defining a mechanism for Secure Shell (SSH) clients and servers to exchange information about supported protocol extensions confidentially after SSH key exchange.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8308.

Table of Contents

1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. The original design of the SSH transport layer [RFC4253] lacks proper extension negotiation. Meanwhile, diverse implementations take steps to ensure that known message types contain no unrecognized information. This makes it difficult for implementations to signal capabilities and negotiate extensions without risking disconnection. This obstacle has been recognized in the process of updating SSH to support RSA signatures using SHA-256 and SHA-512 [RFC8332]. To avoid trial and error as well as authentication penalties, a client must be able to discover public key algorithms a server accepts. This extension mechanism permits this discovery.

This memo updates RFCs 4251, 4252, 4253, and 4254.

1.1. Requirements Terminology

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.

1.2. Wire Encoding Terminology

The wire encoding types in this document -- "byte", "uint32", "string", "boolean", "name-list" -- have meanings as described in [RFC4251].

2. Extension Negotiation Mechanism

2.1. Signaling of Extension Negotiation in SSH_MSG_KEXINIT

Applications implementing this mechanism MUST add one of the following indicator names to the field kex_algorithms in the SSH_MSG_KEXINIT message sent by the application in the first key exchange:

- When acting as server: "ext-info-s"
- When acting as client: "ext-info-c"

The indicator name is added without quotes and MAY be added at any position in the name-list, subject to proper separation from other names as per name-list conventions.

The names are added to the kex_algorithms field because this is one of two name-list fields in SSH_MSG_KEXINIT that do not have a separate copy for each data direction.

The indicator names inserted by the client and server are different to ensure these names will not produce a match and therefore not affect the algorithm chosen in key exchange algorithm negotiation.

The inclusion of textual indicator names is intended to provide a clue for implementers to discover this mechanism.

2.2. Enabling Criteria

If a client or server offers "ext-info-c" or "ext-info-s" respectively, it MUST be prepared to accept an SSH_MSG_EXT_INFO message from the peer.

A server only needs to send "ext-info-s" if it intends to process SSH_MSG_EXT_INFO from the client. A client only needs to send "ext-info-c" if it plans to process SSH_MSG_EXT_INFO from the server.

If a server receives an "ext-info-c", or a client receives an "ext-info-s", it MAY send an SSH_MSG_EXT_INFO message but is not required to do so.

Neither party needs to wait for the other's SSH_MSG_KEXINIT in order to decide whether to send the appropriate indicator in its own SSH_MSG_KEXINIT.

Implementations MUST NOT send an incorrect indicator name for their role. Implementations MAY disconnect if the counterparty sends an incorrect indicator. If "ext-info-c" or "ext-info-s" ends up being negotiated as a key exchange method, the parties MUST disconnect.

2.3. SSH_MSG_EXT_INFO Message

A party that received the "ext-info-c" or "ext-info-s" indicator MAY send the following message:

```
byte       SSH_MSG_EXT_INFO (value 7)
uint32     nr-extensions
repeat the following 2 fields "nr-extensions" times:
  string   extension-name
  string   extension-value (binary)
```

Implementers should pay careful attention to Section 2.5, in particular to the requirement to tolerate any sequence of bytes (including null bytes at any position) in an unknown extension's extension-value.

2.4. Message Order

If a client sends SSH_MSG_EXT_INFO, it MUST send it as the next packet following the client's first SSH_MSG_NEWKEYS message to the server.

If a server sends SSH_MSG_EXT_INFO, it MAY send it at zero, one, or both of the following opportunities:

- As the next packet following the server's first SSH_MSG_NEWKEYS.

  Where clients need information in the server's SSH_MSG_EXT_INFO to authenticate, it is helpful if the server sends its SSH_MSG_EXT_INFO not only as the next packet after SSH_MSG_NEWKEYS, but without delay.

  Clients cannot rely on this because the server is not required to send the message at this time; if sent, it may be delayed by the network. However, if a timely SSH_MSG_EXT_INFO is received, a client can pipeline an authentication request after its SSH_MSG_SERVICE_REQUEST, even when it needs extension information.

- Immediately preceding the server's SSH_MSG_USERAUTH_SUCCESS, as defined in [RFC4252].

  The server MAY send SSH_MSG_EXT_INFO at this second opportunity, whether or not it sent it at the first. A client that sent "ext-info-c" MUST accept a server's SSH_MSG_EXT_INFO at both opportunities but MUST NOT require it.

  This allows a server to reveal support for additional extensions that it was unwilling to reveal to an unauthenticated client. If a server sends a second SSH_MSG_EXT_INFO, this replaces any initial one, and both the client and the server re-evaluate extensions in effect. The server's second SSH_MSG_EXT_INFO is matched against the client's original.

  The timing of the second opportunity is chosen for the following reasons. If the message was sent earlier, it would not allow the server to withhold information until the client has authenticated. If it was sent later, a client that needs information from the second SSH_MSG_EXT_INFO immediately after it authenticates would have no way to reliably know whether to expect the message.

2.5. Interpretation of Extension Names and Values

Each extension is identified by its extension-name and defines the conditions under which the extension is considered to be in effect. Applications MUST ignore unrecognized extension-names.

When it is specified, an extension MAY dictate that, in order to take effect, both parties must include it in their SSH_MSG_EXT_INFO or that it is sufficient for only one party to include it. However, other rules MAY be specified. The relative order in which extensions appear in an SSH_MSG_EXT_INFO message MUST be ignored.

Extension-value fields are interpreted as defined by their respective extension. This field MAY be empty if permitted by the extension. Applications that do not implement or recognize an extension MUST ignore its extension-value, regardless of its size or content. Applications MUST tolerate any sequence of bytes -- including null bytes at any position -- in an unknown extension's extension-value.

The cumulative size of an SSH_MSG_EXT_INFO message is limited only by the maximum packet length that an implementation may apply in accordance with [RFC4253]. Implementations MUST accept well-formed SSH_MSG_EXT_INFO messages up to the maximum packet length they accept.

3. Initially Defined Extensions

3.1. "server-sig-algs"

This extension is sent with the following extension name and value:

```
  string      "server-sig-algs"
  name-list   public-key-algorithms-accepted
```

The name-list type is a strict subset of the string type and is thus permissible as an extension-value. See [RFC4251] for more information.

This extension is sent by the server and contains a list of public key algorithms that the server is able to process as part of a "publickey" authentication request. If a client sends this extension, the server MAY ignore it and MAY disconnect.

In this extension, a server MUST enumerate all public key algorithms it might accept during user authentication. However, early server implementations that do not enumerate all accepted algorithms do

exist. For this reason, a client MAY send a user authentication request using a public key algorithm not included in "server-sig-algs".

A client that wishes to proceed with public key authentication MAY wait for the server's SSH_MSG_EXT_INFO so it can send a "publickey" authentication request with an appropriate public key algorithm, rather than resorting to trial and error.

Servers that implement public key authentication SHOULD implement this extension.

If a server does not send this extension, a client MUST NOT make any assumptions about the server's public key algorithm support, and MAY proceed with authentication requests using trial and error. Note that implementations are known to exist that apply authentication penalties if the client attempts to use an unexpected public key algorithm.

Authentication penalties are applied by servers to deter brute-force password guessing, username enumeration, and other types of behavior deemed suspicious by server administrators or implementers. Penalties may include automatic IP address throttling or blocking, and they may trigger email alerts or auditing.

3.2. "delay-compression"

This extension MAY be sent by both parties as follows:

```
string         "delay-compression"
string:
  name-list    compression_algorithms_client_to_server
  name-list    compression_algorithms_server_to_client
```

The extension-value is a string that encodes two name-lists. The name-lists themselves have the encoding of strings. For example, to indicate a preference for algorithms "foo,bar" in the client-to-server direction and "bar,baz" in the server-to-client direction, a sender encodes the extension-value as follows (including its length):

```
00000016 00000007 666f6f2c626172 00000007 6261722c62617a
```

This same encoding could be sent by either party -- client or server.

This extension allows the server and client to renegotiate compression algorithm support without having to conduct a key re-exchange, which puts new algorithms into effect immediately upon successful authentication.

This extension takes effect only if both parties send it. Name-lists MAY include any compression algorithm that could have been negotiated in SSH_MSG_KEXINIT, except algorithms that define their own delayed compression semantics. This means "zlib,none" is a valid algorithm list in this context, but "zlib@openssh.com" is not.

If both parties send this extension, but the name-lists do not contain a common algorithm in either direction, the parties MUST disconnect in the same way as if negotiation failed as part of SSH_MSG_KEXINIT.

If this extension takes effect, the renegotiated compression algorithm is activated for the very next SSH message after the trigger message:

- Sent by the server, the trigger message is SSH_MSG_USERAUTH_SUCCESS.
- Sent by the client, the trigger message is SSH_MSG_NEWCOMPRESS.

If this extension takes effect, the client MUST send the following message within a reasonable number of outgoing SSH messages after receiving SSH_MSG_USERAUTH_SUCCESS, but not necessarily as the first such outgoing message:

```
byte      SSH_MSG_NEWCOMPRESS (value 8)
```

The purpose of SSH_MSG_NEWCOMPRESS is to avoid a race condition where the server cannot reliably know whether a message sent by the client was sent before or after receiving the server's SSH_MSG_USERAUTH_SUCCESS. For example, clients may send keep-alive messages during logon processing.

As is the case for all extensions unless otherwise noted, the server MAY delay including this extension until its secondary SSH_MSG_EXT_INFO, sent before SSH_MSG_USERAUTH_SUCCESS. This allows the server to avoid advertising compression until the client has authenticated.

If the parties renegotiate compression using this extension in a session where compression is already enabled and the renegotiated algorithm is the same in one or both directions, then the internal compression state MUST be reset for each direction at the time the renegotiated algorithm takes effect.

3.2.1. Awkwardly Timed Key Re-Exchange

A party that has signaled, or intends to signal, support for this extension in an SSH session MUST NOT initiate key re-exchange in that session until either of the following occurs:

- This extension was negotiated, and the party that's about to start key re-exchange already sent its trigger message for compression.
- The party has sent (if server) or received (if client) the message SSH_MSG_USERAUTH_SUCCESS, and this extension was not negotiated.

If a party violates this rule, the other party MAY disconnect.

In general, parties SHOULD NOT start key re-exchange before successful user authentication but MAY tolerate it if not using this extension.

3.2.2. Subsequent Re-Exchange

In subsequent key re-exchanges that unambiguously begin after the compression trigger messages, the compression algorithms negotiated in re-exchange override the algorithms negotiated with this extension.

3.2.3. Compatibility Note: OpenSSH up to Version 7.5

This extension uses a binary extension-value encoding. OpenSSH clients up to and including version 7.5 advertise support to receive SSH_MSG_EXT_INFO but disconnect on receipt of an extension-value containing null bytes. This is an error fixed in OpenSSH version 7.6.

Implementations that wish to interoperate with OpenSSH 7.5 and earlier are advised to check the remote party's SSH version string and omit this extension if an affected version is detected. Affected versions do not implement this extension, so there is no harm in omitting it. The extension SHOULD NOT be omitted if the detected OpenSSH version is 7.6 or higher. This would make it harder for the OpenSSH project to implement this extension in a higher version.

3.3. "no-flow-control"

This extension is sent with the following extension name and value:

```
  string      "no-flow-control"
  string      choice of: "p" for preferred | "s" for supported
```

A party SHOULD send "s" if it supports "no-flow-control" but does not prefer to enable it. A party SHOULD send "p" if it prefers to enable the extension if the other party supports it. Parties MAY disconnect if they receive a different extension value.

For this extension to take effect, the following must occur:

- This extension MUST be sent by both parties.
- At least one party MUST have sent the value "p" (preferred).

If this extension takes effect, the "initial window size" fields in SSH_MSG_CHANNEL_OPEN and SSH_MSG_CHANNEL_OPEN_CONFIRMATION, as defined in [RFC4254], become meaningless. The values of these fields MUST be ignored, and a channel behaves as if all window sizes are infinite. Neither side is required to send any SSH_MSG_CHANNEL_WINDOW_ADJUST messages, and if received, such messages MUST be ignored.

This extension is intended for, but not limited to, use by file transfer applications that are only going to use one channel and for which the flow control provided by SSH is an impediment, rather than a feature.

Implementations MUST refuse to open more than one simultaneous channel when this extension is in effect. Nevertheless, server implementations SHOULD support clients opening more than one non-simultaneous channel.

3.3.1. Prior "No Flow Control" Practice

Before this extension, some applications would simply not implement SSH flow control, sending an initial channel window size of 2^32 - 1. Applications SHOULD NOT do this for the following reasons:

- It is plausible to transfer more than 2^32 bytes over a channel. Such a channel will hang if the other party implements SSH flow control according to [RFC4254].

- Implementations that cannot handle large channel window sizes exist, and they can exhibit non-graceful behaviors, including disconnect.

3.4. "elevation"

The terms "elevation" and "elevated" refer to an operating system mechanism where an administrator's logon session is associated with two security contexts: one limited and one with administrative rights. To "elevate" such a session is to activate the security context with full administrative rights. For more information about this mechanism on Windows, see [WINADMIN] and [WINTOKEN].

This extension MAY be sent by the client as follows:

```
string      "elevation"
string      choice of: "y" | "n" | "d"
```

A client sends "y" to indicate its preference that the session should be elevated; "n" to not be elevated; and "d" for the server to use its default behavior. The server MAY disconnect if it receives a different extension value. If a client does not send the "elevation" extension, the server SHOULD act as if "d" was sent.

If a client has included this extension, then after authentication, a server that supports this extension SHOULD indicate to the client whether elevation was done by sending the following global request:

```
byte        SSH_MSG_GLOBAL_REQUEST
string      "elevation"
boolean     want reply = false
boolean     elevation performed
```

Clients that implement this extension help reduce attack surface for Windows servers that handle administrative logins. Where clients do not support this extension, servers must elevate sessions to allow full access by administrative users always. Where clients support this extension, sessions can be created without elevation unless requested.

4. IANA Considerations

4.1. Additions to Existing Registries

IANA has added the following entries to the "Message Numbers" registry [IANA-M] under the "Secure Shell (SSH) Protocol Parameters" registry [RFC4250]:

Value	Message ID	Reference
7	SSH_MSG_EXT_INFO	RFC 8308
8	SSH_MSG_NEWCOMPRESS	RFC 8308

IANA has also added the following entries to the "Key Exchange Method Names" registry [IANA-KE]:

Method Name	Reference	Note
ext-info-s	RFC 8308	Section 2
ext-info-c	RFC 8308	Section 2

4.2. New Registry: Extension Names

Also under the "Secure Shell (SSH) Protocol Parameters" registry, IANA has created a new "Extension Names" registry, with the following initial content:

Extension Name	Reference	Note
server-sig-algs	RFC 8308	Section 3.1
delay-compression	RFC 8308	Section 3.2
no-flow-control	RFC 8308	Section 3.3
elevation	RFC 8308	Section 3.4

4.2.1. Future Assignments to Extension Names Registry

Names in the "Extension Names" registry MUST follow the conventions for names defined in [RFC4250], Section 4.6.1.

Requests for assignments of new non-local names in the "Extension Names" registry (i.e., names not including the '@' character) MUST be done using the IETF Review policy, as described in [RFC8126].

5. Security Considerations

Security considerations are discussed throughout this document. This document updates the SSH protocol as defined in [RFC4251] and related documents. The security considerations of [RFC4251] apply.

6. References

6.1. 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>.

[RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <https://www.rfc-editor.org/info/rfc4251>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/info/rfc4252>.

[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>.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <https://www.rfc-editor.org/info/rfc4254>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

6.2. Informative References

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

[IANA-M] IANA, "Message Numbers", <https://www.iana.org/assignments/ssh-parameters/>.

[RFC8332] Bider, D., "Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol", RFC 8332, DOI 10.17487/RFC8332, March 2018, <https://www.rfc-editor.org/info/rfc8332>.

[WINADMIN] Microsoft, "How to launch a process as a Full Administrator when UAC is enabled?", March 2013, <https://blogs.msdn.microsoft.com/winsdk/2013/03/22/how-to-launch-a-process-as-a-full-administrator-when-uac-is-enabled/>.

[WINTOKEN] Microsoft, "TOKEN_ELEVATION_TYPE enumeration", <https://msdn.microsoft.com/en-us/library/windows/desktop/bb530718.aspx>.

Acknowledgments

Thanks to Markus Friedl and Damien Miller for comments and initial implementation. Thanks to Peter Gutmann, Roumen Petrov, Mark D. Baushke, Daniel Migault, Eric Rescorla, Matthew A. Miller, Mirja Kuehlewind, Adam Roach, Spencer Dawkins, Alexey Melnikov, and Ben Campbell for reviews and feedback.

Author's Address

Denis Bider
Bitvise Limited
4105 Lombardy Court
Colleyville, TX 76034
United States of America

Email: ietf-ssh3@denisbider.com
URI: https://www.bitvise.com/