Internet-Draft
Updates: 4252, 4253, 4254 (if approved)
Intended status: Standards Track
Expires: February 22, 2018

D. Bider
Bitvise Limited

August 22, 2017

# Extension Negotiation in Secure Shell (SSH)
# draft-ietf-curdle-ssh-ext-info-12.txt

## Abstract

This memo updates RFC 4252, RFC 4253, and RFC 4254 to define a mechanism for SSH clients and servers to exchange information about supported protocol extensions confidentially after SSH key exchange.

## Status

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

## Copyright

## 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 relationship with [SSH-RSA-SHA2], where the need arises for a client to discover public key algorithms a server accepts, to avoid authentication penalties and trial-and-error.

This memo updates RFC 4252, RFC 4253, and RFC 4254.

### 1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

### 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 KEXINIT

Applications implementing this mechanism MUST add to the field "kex_algorithms", in their KEXINIT packet sent for the first key exchange, one of the following indicator names:

- 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 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 KEXINIT in order to decide whether to send the appropriate indicator in its own KEXINIT.

Implementations MUST NOT send an incorrect indicator name for their role. Implementations MAY disconnect if the counter-party 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)
```

This message is sent immediately after SSH_MSG_NEWKEYS, without delay. This allows a client to pipeline an authentication request after its SSH_MSG_SERVICE_REQUEST, even when this needs extension information.

Receivers MUST tolerate any sequence of bytes; including null bytes at any position; in an unknown extension's extension-value.

## 2.4. Server's Secondary SSH_MSG_EXT_INFO

If the client sent "ext-info-c", the server MAY send zero, one, or two EXT_INFO messages. The first opportunity for the server's EXT_INFO is after the server's NEWKEYS, as above. The second opportunity is just before (*) SSH_MSG_USERAUTH_SUCCESS, as defined in [RFC4252]. The server MAY send EXT_INFO at the second opportunity, whether or not it sent it at the first. A client that sent "ext-info-c" MUST accept a server's 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 subsequent SSH_MSG_EXT_INFO, this replaces any initial one, and both the client and the server re-evaluate extensions in effect. The server's last EXT_INFO is matched against the client's original.

(*) The message MUST be sent at this point for the following reasons: if it 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 EXT_INFO immediately after successful authentication would have no way of reliably knowing whether there will be a second EXT_INFO or not.

### 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.

An extension MAY dictate, where it is specified, that in order to take effect, both parties must include it in their EXT_INFO; or it MAY be sufficient that only one party includes it; or other rules MAY be specified. The relative order in which extensions appear in an EXT_INFO message MUST be ignored by default; but an extension MAY specify that the order matters for that extension, in a specific way.

Extension-value fields are interpreted as defined by their respective extension. An extension-value field MAY be empty if so permitted by the extension. Applications that do not implement or recognize a particular extension MUST ignore the associated extension-value field, regardless of its size or content. In particular, 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, there exist early server implementations which do not enumerate all accepted algorithms. 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.

### 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
```

This extension allows the server and client to renegotiate compression algorithm support without having to conduct a key re-exchange, putting 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 shortly after receiving SSH_MSG_USERAUTH_SUCCESS:

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

The purpose of 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 USERAUTH_SUCCESS.

As with all extensions, the server MAY delay including this extension until its secondary SSH_MSG_EXT_INFO, sent before USERAUTH_SUCCESS. This allows the server to avoid advertising compression support until the client has been authenticated.

If the parties re-negotiate compression using this extension in a session where compression is already enabled; and the re-negotiated algorithm is the same in one or both directions; then the internal compression state MUST be reset for each direction at the time the re-negotiated 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.

### 3.3. "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.

To take effect, 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, 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. Implementation Note: 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 entirely within the realm of possibility to transfer more than 2^32 bytes over a channel. The channel will then hang if the other party implements SSH flow control according to [RFC4254].

- There exist implementations which cannot handle such large channel window sizes, and will exhibit non-graceful behaviors, including disconnection.

### 3.4. "elevation"

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.

The terms "elevation" and "elevated" refer to an operating system mechanism where an administrator user'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 also [WINADMIN] and [WINTOKEN].

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
```

## 4. IANA Considerations

### 4.1. Additions to existing tables

IANA is requested to insert the following entries into the table Message Numbers [IANA-M] under Secure Shell (SSH) Protocol Parameters [RFC4250]:

| Value | Message ID | Reference |
|---|---|---|
| 7 | SSH_MSG_EXT_INFO | [this document] |
| 8 | SSH_MSG_NEWCOMPRESS | [this document] |

IANA is requested to insert the following entries into the table Key Exchange Method Names [IANA-KE]:

| Method Name | Reference | Note |
|---|---|---|
| ext-info-s | [this document] | Section 2.2 |
| ext-info-c | [this document] | Section 2.2 |

### 4.2. New table: Extension Names

Also under Secure Shell (SSH) Protocol Parameters, IANA is requested to create a new table, Extension Names, with initial content:

| Extension Name | Reference | Note |
|---|---|---|
| server-sig-algs | [this document] | Section 3.1 |
| delay-compression | [this document] | Section 3.2 |
| no-flow-control | [this document] | Section 3.3 |
| elevation | [this document] | Section 3.4 |

#### 4.2.1. Future Assignments to Extension Names

Names in the Extension Names table 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 table (i.e. names not including the '@' character) MUST be done through the IETF CONSENSUS method, 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, March 1997.

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.

[RFC4251] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.

[RFC8126] Cotton, M., Leiba, B. and Narten, T., "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, June 2017.

## 6.2. Informative References

[SSH-RSA-SHA2] Bider, D., "Use of RSA Keys with SHA-2 256 and 512 in Secure Shell (SSH)", draft-ietf-curdle-rsa-sha2-10.txt, August 2017, <https://tools.ietf.org/html/draft-ietf-curdle-rsa-sha2-10>.

[IANA-M] "Secure Shell (SSH) Protocol Parameters", <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml#ssh-parameters-1>.

[IANA-KE] "Secure Shell (SSH) Protocol Parameters", <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml#ssh-parameters-16>.

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

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

bb530718.aspx>.

## Author's Address

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

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

## 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, and Matthew A. Miller for review and feedback.