Network Working Group
Request for Comments: 4819
Category: Standards Track

J. Galbraith
J. Van Dyke
VanDyke Software
J. Bright
Silicon Circus

March 2007

Secure Shell Public Key Subsystem

Status of This Memo

Copyright Notice

Abstract

Secure Shell defines a user authentication mechanism that is based on public keys, but does not define any mechanism for key distribution. No common key management solution exists in current implementations. This document describes a protocol that can be used to configure public keys in an implementation-independent fashion, allowing client software to take on the burden of this configuration.

The Public Key Subsystem provides a server-independent mechanism for clients to add public keys, remove public keys, and list the current public keys known by the server. Rights to manage public keys are specific and limited to the authenticated user.

A public key may also be associated with various restrictions, including a mandatory command or subsystem.

Table of Contents

1. Introduction

Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network. Secure Shell defines a user authentication mechanism that is based on public keys, but does not define any mechanism for key distribution. Common practice is to authenticate once with password authentication and transfer the public key to the server. However, to date no two implementations use the same mechanism to configure a public key for use.

This document describes a subsystem that can be used to configure public keys in an implementation-independent fashion. This approach allows client software to take on the burden of this configuration. The Public Key Subsystem protocol is designed for extreme simplicity in implementation. It is not intended as a Public Key Infrastructure for X.509 Certificates (PKIX) replacement.

The Secure Shell Public Key Subsystem has been designed to run on top of the Secure Shell transport layer [2] and user authentication protocols [3]. It provides a simple mechanism for the client to manage public keys on the server.

This document should be read only after reading the Secure Shell architecture [1] and Secure Shell connection [4] documents.

This protocol is intended to be used from the Secure Shell Connection Protocol [4] as a subsystem, as described in the section "Starting a Shell or a Command". The subsystem name used with this protocol is "publickey".

This protocol requires that the user be able to authenticate in some fashion before it can be used. If password authentication is used, servers SHOULD provide a configuration option to disable the use of password authentication after the first public key is added.

2. 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 RFC 2119 [5].

3. Public Key Subsystem Overview

The Public Key Subsystem provides a server-independent mechanism for clients to add public keys, remove public keys, and list the current public keys known by the server. The subsystem name is "publickey".

The public keys added, removed, and listed using this protocol are specific and limited to those of the authenticated user.

The operations to add, remove, and list the authenticated user's public keys are performed as request packets sent to the server. The server sends response packets that indicate success or failure as well as provide specific response data.

The format of public key blobs are detailed in Section 6.6, "Public Key Algorithms" of the SSH Transport Protocol document [2].

3.1. Opening the Public Key Subsystem

The Public Key Subsystem is started by a client sending an SSH_MSG_CHANNEL_REQUEST over an existing session's channel.

The details of how a session is opened are described in the SSH Connection Protocol document [4] in the section "Opening a Session".

To open the Public Key Subsystem, the client sends:

```
     byte      SSH_MSG_CHANNEL_REQUEST
     uint32    recipient channel
     string    "subsystem"
     boolean   want reply
     string    "publickey"
```

Client implementations SHOULD reject this request; it is normally sent only by the client.

If want reply is TRUE, the server MUST respond with SSH_MSG_CHANNEL_SUCCESS if the Public Key Subsystem was successfully started, or SSH_MSG_CHANNEL_FAILURE if the server failed to start or does not support the Public Key Subsystem.

The server SHOULD respond with SSH_MSG_CHANNEL_FAILURE if the user is not allowed access to the Public Key Subsystem (for example, because the user authenticated with a restricted public key).

It is RECOMMENDED that clients request and check the reply for this request.

3.2. Requests and Responses

All Public Key Subsystem requests and responses are sent in the following form:

```
     uint32    length
     string    name
     ... request/response specific data follows
```

The length field describes the length of the name field and of the request/response-specific data, but does not include the length of the length field itself. The client MUST receive acknowledgement of each request prior to sending a new request.

The version packet, as well as all requests and responses described in Section 4, are a description of the 'name' field and the data part of the packet.

3.3. The Status Message

A request is acknowledged by sending a status packet. If there is data in response to the request, the status packet is sent after all data has been sent.

```
     string    "status"
     uint32    status code
     string    description [7]
     string    language tag [6]
```

A status message MUST be sent for any unrecognized packets, and the request SHOULD NOT close the subsystem.

3.3.1. Status Codes

The status code gives the status in a more machine-readable format (suitable for localization), and can have the following values:

SSH_PUBLICKEY_SUCCESS	0
SSH_PUBLICKEY_ACCESS_DENIED	1
SSH_PUBLICKEY_STORAGE_EXCEEDED	2
SSH_PUBLICKEY_VERSION_NOT_SUPPORTED	3
SSH_PUBLICKEY_KEY_NOT_FOUND	4
SSH_PUBLICKEY_KEY_NOT_SUPPORTED	5
SSH_PUBLICKEY_KEY_ALREADY_PRESENT	6
SSH_PUBLICKEY_GENERAL_FAILURE	7
SSH_PUBLICKEY_REQUEST_NOT_SUPPORTED	8
SSH_PUBLICKEY_ATTRIBUTE_NOT_SUPPORTED	9

If a request completed successfully, the server MUST send the status code SSH_PUBLICKEY_SUCCESS. The meaning of the failure codes is as implied by their names.

3.4. The Version Packet

Both sides MUST start a connection by sending a version packet that indicates the version of the protocol they are using.

```
string "version"
uint32 protocol-version-number
```

This document describes version 2 of the protocol. Version 1 was used by an early draft of this document. The version number was incremented after changes in the handling of status packets.

Both sides send the highest version that they implement. The lower of the version numbers is the version of the protocol to use. If either side can't support the lower version, it should close the subsystem and notify the other side by sending an SSH_MSG_CHANNEL_CLOSE message. Before closing the subsystem, a status message with the status SSH_PUBLICKEY_VERSION_NOT_SUPPORTED SHOULD be sent. Note that, normally, status messages are only sent by the server (in response to requests from the client). This is the only occasion on which the client sends a status message.

Both sides MUST wait to receive this version before continuing. The "version" packet MUST NOT be sent again after this initial exchange. The SSH_PUBLICKEY_VERSION_NOT_SUPPORTED status code must not be sent in response to any other request.

Implementations MAY use the first 15 bytes of the version packet as a "magic cookie" to avoid processing spurious output from the user's shell (as described in Section 6.5 of [4]). These bytes will always be:

0x00 0x00 0x00 0x0F 0x00 0x00 0x00 0x07 0x76 0x65 0x72 0x73 0x69 0x6F 0x6E

4. Public Key Subsystem Operations

The Public Key Subsystem currently defines four operations: add, remove, list, and listattributes.

4.1. Adding a Public Key

If the client wishes to add a public key, the client sends:

```
     string    "add"
     string    public key algorithm name
     string    public key blob
     boolean   overwrite
     uint32    attribute-count
      string    attrib-name
      string    attrib-value
      bool      critical
     repeated attribute-count times
```

The server MUST attempt to store the public key for the user in the appropriate location so the public key can be used for subsequent public key authentications. If the overwrite field is false and the specified key already exists, the server MUST return SSH_PUBLICKEY_KEY_ALREADY_PRESENT. If the server returns this, the client SHOULD provide an option to the user to overwrite the key. If the overwrite field is true and the specified key already exists, but cannot be overwritten, the server MUST return SSH_PUBLICKEY_ACCESS_DENIED.

Attribute names are defined following the same scheme laid out for algorithm names in [1]. If the server does not implement a critical attribute, it MUST fail the add, with the status code SSH_PUBLICKEY_ATTRIBUTE_NOT_SUPPORTED. For the purposes of a critical attribute, mere storage of the attribute is not sufficient -- rather, the server must understand and implement the intent of the attribute.

The following attributes are currently defined:

"comment"

The value of the comment attribute contains user-specified text about the public key. The server SHOULD make every effort to preserve this value and return it with the key during any subsequent list operation. The server MUST NOT attempt to interpret or act upon the content of the comment field in any way. The comment attribute must be specified in UTF-8 format [7].

The comment field is useful so the user can identify the key without resorting to comparing its fingerprint. This attribute SHOULD NOT be critical.

"comment-language"

If this attribute is specified, it MUST immediately follow a "comment" attribute and specify the language for that attribute [6]. The client MAY specify more than one comment if it additionally specifies a different language for each of those comments. The server SHOULD attempt to store each comment with its language attribute. This attribute SHOULD NOT be critical.

"command-override"

"command-override" specifies a command to be executed when this key is in use. The command should be executed by the server when it receives an "exec" or "shell" request from the client, in place of the command or shell which would otherwise have been executed as a result of that request. If the command string is empty, both "exec" and "shell" requests should be denied. If no "command-override" attribute is specified, all "exec" and "shell" requests should be permitted (as long as they satisfy other security or authorization checks the server may perform). This attribute SHOULD be critical.

"subsystem"

"subsystem" specifies a comma-separated list of subsystems that may be started (using a "subsystem" request) when this key is in use. This attribute SHOULD be critical. If the value is empty, no subsystems may be started. If the "subsystem" attribute is not specified, no restrictions are placed on which subsystems may be started when authenticated using this key.

"x11"

"x11" specifies that X11 forwarding may not be performed when this key is in use. The attribute-value field SHOULD be empty for this attribute. This attribute SHOULD be critical.

"shell"

"shell" specifies that session channel "shell" requests should be denied when this key is in use. The attribute-value field SHOULD be empty for this attribute. This attribute SHOULD be critical.

"exec"

"exec" specifies that session channel "exec" requests should be denied when this key is in use. The attribute-value field SHOULD be empty for this attribute. This attribute SHOULD be critical.

"agent"

"agent" specifies that session channel "auth-agent-req" requests should be denied when this key is in use. The attribute-value field SHOULD be empty for this attribute. This attribute SHOULD be critical.

"env"

"env" specifies that session channel "env" requests should be denied when this key is in use. The attribute-value field SHOULD be empty for this attribute. This attribute SHOULD be critical.

"from"

"from" specifies a comma-separated list of hosts from which the key may be used. If a host not in this list attempts to use this key for authorization purposes, the authorization attempt MUST be denied. The server SHOULD make a log entry regarding this. The server MAY provide a method for administrators to disallow the appearance of a host in this list. The server should use whatever method is appropriate for its platform to identify the host -- e.g., for IP-based networks, checking the IP address or performing a reverse DNS lookup. For IP-based networks, it is anticipated that each element of the "from" parameter will take the form of a specific IP address or hostname.

"port-forward"

"port-forward" specifies that no "direct-tcpip" requests should be accepted, except those to hosts specified in the comma-separated list supplied as a value to this attribute. If the value of this attribute is empty, all "direct-tcpip" requests should be refused when using this key. This attribute SHOULD be critical.

"reverse-forward"

"reverse-forward" specifies that no "tcpip-forward" requests should be accepted, except for the port numbers in the comma-separated list supplied as a value to this attribute. If the value of this attribute is empty, all "tcpip-forward" requests should be refused when using this key. This attribute SHOULD be critical.

In addition to the attributes specified by the client, the server MAY provide a method for administrators to enforce certain attributes compulsorily.

4.2. Removing a Public Key

If the client wishes to remove a public key, the client sends:

```
string    "remove"
string    public key algorithm name
string    public key blob
```

The server MUST attempt to remove the public key for the user from the appropriate location, so that the public key cannot be used for subsequent authentications.

4.3. Listing Public Keys

If the client wishes to list the known public keys, the client sends:

```
string    "list"
```

The server will respond with zero or more of the following responses:

```
string    "publickey"
string    public key algorithm name
string    public key blob
uint32    attribute-count
 string    attrib-name
 string    attrib-value
repeated attribute-count times
```

There is no requirement that the responses be in any particular order. Whilst some server implementations may send the responses in some order, client implementations should not rely on responses being in any order.

Following the last "publickey" response, a status packet MUST be sent.

Implementations SHOULD support this request.

4.4. Listing Server Capabilities

If the client wishes to know which key attributes the server supports, it sends:

```
string    "listattributes"
```

The server will respond with zero or more of the following responses:

```
      string    "attribute"
      string    attribute name
      boolean   compulsory
```

The "compulsory" field indicates whether this attribute will be compulsorily applied to any added keys (irrespective of whether the attribute has been specified by the client) due to administrative settings on the server. If the server does not support administrative settings of this nature, it MUST return false in the compulsory field. An example of use of the "compulsory" attribute would be a server with a configuration file specifying that the user is not permitted shell access. Given this, the server would return the "shell" attribute, with "compulsory" marked true. Whatever attributes the user subsequently asked the server to apply to their key, the server would also apply the "shell" attribute, rendering it impossible for the user to use a shell.

Following the last "attribute" response, a status packet MUST be sent.

An implementation MAY choose not to support this request.

5. Security Considerations

This protocol assumes that it is run over a secure channel and that the endpoints of the channel have been authenticated. Thus, this protocol assumes that it is externally protected from network-level attacks.

This protocol provides a mechanism that allows client authentication data to be uploaded and manipulated. It is the responsibility of the server implementation to enforce any access controls that may be required to limit the access allowed for any particular user (the user being authenticated externally to this protocol, typically using the SSH User Authentication Protocol [3]). In particular, it is possible for users to overwrite an existing key on the server with this protocol, whilst at the same time specifying fewer restrictions for the new key than were previously present. Servers should take care that when doing this, clients are not able to override presets from the server's administrator.

This protocol requires the client to assume that the server will correctly implement and observe attributes applied to keys. Implementation errors in the server could cause clients to authorize keys for access they were not intended to have, or to apply fewer restrictions than were intended.

6. IANA Considerations

This section contains conventions used in naming the namespaces, the initial state of the registry, and instructions for future assignments.

6.1. Registrations

Consistent with Section 4.9.5 of [8], this document makes the following registration:

The subsystem name "publickey".

6.2. Names

In the following sections, the values for the namespaces are textual. The conventions and instructions to the IANA for future assignments are given in this section. The initial assignments are given in their respective sections.

6.2.1. Conventions for Names

All names registered by the IANA in the following sections MUST be printable US-ASCII strings, and MUST NOT contain the characters at-sign ("@"), comma (","), or whitespace or control characters (ASCII codes 32 or less). Names are case-sensitive, and MUST NOT be longer than 64 characters.

A provision is made here for locally extensible names. The IANA will not register and will not control names with the at-sign in them. Names with the at-sign in them will have the format of "name@domainname" (without the double quotes) where the part preceding the at-sign is the name. The format of the part preceding the at-sign is not specified; however, these names MUST be printable US-ASCII strings, and MUST NOT contain the comma character (","), or whitespace, or control characters (ASCII codes 32 or less). The part following the at-sign MUST be a valid, fully qualified Internet domain name [10] controlled by the person or organization defining the name. Names are case-sensitive, and MUST NOT be longer than 64 characters. It is up to each domain how it manages its local namespace. It has been noted that these names resemble STD 11 [9] email addresses. This is purely coincidental and actually has nothing to do with STD 11 [9]. An example of a locally defined name is "our-attribute@example.com" (without the double quotes).

6.2.2. Future Assignments of Names

Requests for assignments of new Names MUST be done through the IETF Consensus method as described in [11].

6.3. Public Key Subsystem Request Names

The following table lists the initial assignments of Public Key Subsystem Request names.

Request Name
version
add
remove
list
listattributes

6.4. Public Key Subsystem Response Names

The following table lists the initial assignments of Public Key Subsystem Response names.

Response Name
version
status
publickey
attribute

6.5. Public Key Subsystem Attribute Names

Attributes are used to define properties or restrictions for public keys. The following table lists the initial assignments of Public Key Subsystem Attribute names.

Attribute Name
comment
comment-language
command-override
subsystem
x11
shell
exec
agent
env
from
port-forward
reverse-forward

6.6. Public Key Subsystem Status Codes

The status code is a byte value, describing the status of a request.

6.6.1. Conventions

Status responses have status codes in the range 0 to 255. These numbers are allocated as follows. Of these, the range 192 to 255 is reserved for use by local, private extensions.

6.6.2. Initial Assignments

The following table identifies the initial assignments of the Public Key Subsystem status code values.

Status code	Value	Reference
SSH_PUBLICKEY_SUCCESS	0	
SSH_PUBLICKEY_ACCESS_DENIED	1	
SSH_PUBLICKEY_STORAGE_EXCEEDED	2	
SSH_PUBLICKEY_VERSION_NOT_SUPPORTED	3	
SSH_PUBLICKEY_KEY_NOT_FOUND	4	
SSH_PUBLICKEY_KEY_NOT_SUPPORTED	5	
SSH_PUBLICKEY_KEY_ALREADY_PRESENT	6	
SSH_PUBLICKEY_GENERAL_FAILURE	7	
SSH_PUBLICKEY_REQUEST_NOT_SUPPORTED	8	
SSH_PUBLICKEY_ATTRIBUTE_NOT_SUPPORTED	9	

6.6.3. Future Assignments

Requests for assignments of new status codes in the range of 0 to 191 MUST be done through the Standards Action method as described in [11].

The IANA will not control the status code range of 192 through 255. This range is for private use.

7. References

7.1. Normative References

[1] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

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

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

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

[5] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[6] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 4646, September 2006.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

7.2. Informative References

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

[9] Crocker, D., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[10] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[11] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

8. Acknowledgements

Brent McClure contributed to the writing of this document.

Authors' Addresses

Joseph Galbraith
VanDyke Software
4848 Tramway Ridge Blvd
Suite 101
Albuquerque, NM 87111
US

Phone: +1 505 332 5700
EMail: galb@vandyke.com

Jeff P. Van Dyke
VanDyke Software
4848 Tramway Ridge Blvd
Suite 101
Albuquerque, NM 87111
US

Phone: +1 505 332 5700
EMail: jpv@vandyke.com

Jon Bright
Silicon Circus
24 Jubilee Road
Chichester, West Sussex PO19 7XB
UK

Phone: +49 172 524 0521
EMail: jon@siliconcircus.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.