Network Working Group
Internet-Draft
Expires: January 14, 2006

B. Harris

July 13, 2005

Rivest-Shamir-Adleman (RSA) key exchange for the Secure Shell (SSH) Transport Layer Protocol
draft-harris-ssh-rsa-kex-03

Status of this Memo

Copyright Notice

Abstract

This memo describes a key-exchange method for the Secure Shell (SSH) protocol based on Rivest-Shamir-Adleman (RSA) public-key encryption. It uses much less client CPU time than the Diffie-Hellman algorithm specified as part of the core protocol, and hence is particularly suitable for slow client systems.

1. Introduction

Secure Shell (SSH) [I-D.ietf-secsh-architecture] is a secure remote-login protocol. The core protocol uses Diffie-Hellman key exchange. On slow CPUs, this key exchange can take tens of seconds to complete, which can be irritating for the user. A previous version of the SSH protocol, described in [SSH1] uses a key-exchange method based on Rivest-Shamir-Adleman (RSA) public-key encryption, which consumes an order of magnitude less CPU time on the client, and hence is particularly suitable for slow client systems such as mobile devices. This memo describes a key-exchange mechanism for the version of SSH described in [I-D.ietf-secsh-architecture] which is similar to that used by the older version, and about as fast, while retaining the security advantages of the newer protocol.

2. Conventions Used in this Document

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].

3. Overview

The RSA key-exchange method consists of three messages. First, the server sends to the client an RSA public key, K_T, to which the server holds the private key. This may be a transient key generated solely for this SSH connection, or it may be re-used for several connections. The client generates a string of random bytes, K, encrypts it using K_T, and sends the result back to the server, which decrypts it. The client and server each hash K, K_T, and the various key-exchange parameters to generate the exchange hash, H, which is used to generate the encryption keys for the session, and the server signs H with its host key and sends the signature to the client. The client then verifies the host key as described in [I-D.ietf-secsh-transport].

This method provides explicit server identification as defined in section 7 of [I-D.ietf-secsh-transport]. It requires a signature-capable host key.

4. Details

The RSA key exchange method has the following parameters, which are

defined by the method name:

```
    HASH     hash algorithm for calculating exchange hash etc.
    HLEN     output length of HASH in bits
    MINKLEN  minimum transient RSA modulus length in bits
```


The method uses the following messages.

First, the server sends:

```
    byte      SSH_MSG_KEXRSA_PUBKEY
    string    K_T, transient RSA public key
    string    server public host key and certificates (K_S)
```

The key K_T is encoded according to the "ssh-rsa" scheme described in section 6.6 of [I-D.ietf-secsh-transport]. Note that unlike an "ssh-rsa" host key, K_T is only used for encryption, and not for signature. The modulus of K_T MUST be at least MINKLEN bits long.

The client generates a random integer, K, in the range 0 <= K < 2^(KLEN-2HLEN-49), where KLEN is the length of the modulus of K_T, in bits. The client then uses K_T to encrypt:

```
    mpint     K, the shared secret
```

The encryption is performed according to the RSAES-OAEP scheme of [RFC3447], with a mask generation function of MGF1-with-HASH, a hash of HASH, and an empty label. See Appendix A for a proof that the encoding of K is always short enough to be thus encrypted. Having performed the encryption, the client sends:

```
    byte      SSH_MSG_KEXRSA_SECRET
    string    RSAES-OAEP-ENCRYPT(K_T, K)
```

Note that the last stage of RSAES-OAEP-ENCRYPT is to encode an integer as an octet-string using the I2OSP primitive of [RFC3447]. This, combined with encoding the result as an SSH "string", gives a result which is similar, but not identical, to the SSH "mpint" encoding applied to that integer. This is the same encoding as is used by "ssh-rsa" signatures in [I-D.ietf-secsh-transport].

The server decrypts K. If a decryption error occurs, the server

SHOULD send SSH_MESSAGE_DISCONNECT with a reason code of SSH_DISCONNECT_KEY_EXCHANGE_FAILED and MUST disconnect. Otherwise, the server responds with:

```
byte      SSH_MSG_KEXRSA_DONE
string    signature of H with host key
```


The hash H is computed as the HASH hash of the concatenation of the following:

```
string    V_C, the client's version string (CR and NL excluded)
string    V_S, the server's version string (CR and NL excluded)
string    I_C, the payload of the client's SSH_MSG_KEXINIT
string    I_S, the payload of the server's SSH_MSG_KEXINIT
string    K_S, the host key
string    K_T, the transient RSA key
string    RSAES_OAEP_ENCRYPT(K_T, K), the encrypted secret
mpint     K, the shared secret
```

This value is called the exchange hash, and it is used to authenticate the key exchange. The exchange hash SHOULD be kept secret.

The signature algorithm MUST be applied over H, not the original data. Most signature algorithms include hashing and additional padding - for example, "ssh-dss" specifies SHA-1 hashing. In that case, the data is first hashed with HASH to compute H, and H is then hashed with SHA-1 as part of the signing operation.

5. rsa1024-sha1-draft-03@putty.projects.tartarus.org

The "rsa1024-sha1-draft-03@putty.projects.tartarus.org" method specifies RSA key exchange as described above with the following parameters:

```
HASH     SHA-1, as defined in [RFC3174]
```

```
HLEN     160
MINKLEN  1024
```

6. rsa2048-sha256-draft-03@putty.projects.tartarus.org

The "rsa2048-sha256-draft-03@putty.projects.tartarus.org" method specifies RSA key exchange as described above with the following parameters:

```
HASH     SHA-256, as defined in [FIPS-180-2]
HLEN     256
MINKLEN  2048
```


7. Message numbers

The following message numbers are defined:

```
SSH_MSG_KEXRSA_PUBKEY  30
SSH_MSG_KEXRSA_SECRET  31
SSH_MSG_KEXRSA_DONE    32
```

8. Security Considerations

The security considerations in [I-D.ietf-secsh-architecture] apply.

If the RSA private key generated by the server is revealed then the session key is revealed. The server should thus arrange to erase this from memory as soon as it is no longer required. If the same RSA key is used for multiple SSH connections, an attacker who can find the private key (either by factorising the public key or by other means) will gain access to all of the sessions which used that key. As a result, servers SHOULD use each RSA key for as few key exchanges as possible.

[RFC3447] recommends that RSA keys used with RSAES-OAEP not be used with other schemes, or with RSAES-OAEP using a different hash function. In particular, this means that K_T should not be used as a host key, or as a server key in earlier versions of the SSH protocol.

The random data, K, generated by the client, is the only secret shared by client and server, so its entropy directly determines the security of the session against eavesdropping.

The size of transient key used should be sufficient to protect the encryption and integrity keys generated by the key exchange method. For recommendations on this, see [RFC3766]. The strength of RSAES-OAEP is in part dependent on the hash function it uses. [RFC3447] suggests using a hash with an output length of twice the security level required, so SHA-1 is appropriate for applications requiring up to 80 bits of security, and SHA-512 for those requiring up to 256 bits.

Unlike the Diffie-Hellman key exchange method defined by [I-D.ietf-secsh-transport], this method allows the client to fully determine the shared secret, K. This is believed not to be significant, since K is only ever used when hashed with data provided in part by the server (usually in the form of the exchange hash, H). If an extension to SSH were to use K directly and to assume that it had been generated by Diffie-Hellman key exchange, this could produce a security weakness. Protocol extensions using K directly should be

viewed with extreme suspicion.

9. IANA Considerations

This document has no actions for IANA.

10. Acknowledgments

The author acknowledges the assistance of Simon Tatham with the design of this key exchange method.

The text of this document is derived in part from [I-D.ietf-secsh-transport].

11. References

11.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[I-D.ietf-secsh-architecture] Ylonen, T. and C. Lonvick, "SSH Protocol Architecture", draft-ietf-secsh-architecture-22 (work in progress), March 2005.

[I-D.ietf-secsh-transport] Lonvick, C., "SSH Transport Layer Protocol", draft-ietf-secsh-transport-24 (work in progress), March 2005.

[FIPS-180-2] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", FIPS PUB 180-2, August 2002.

11.2 Informative References

[SSH1] Ylonen, T., "SSH -- Secure Login Connections over the Internet", 6th USENIX Security Symposium, pp. 37-42, July 1996.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.

Author's Address

Ben Harris

37 Milton Road
CAMBRIDGE CB4 1XA
GB

Email: bjh21@bjh21.me.uk

Appendix A. On the size of K

The requirements on the size of K are intended to ensure that it's always possible to encrypt it under K_T. The mpint encoding of K requires a leading zero bit, padding to a whole number of bytes, and a four-byte length field, giving a maximum length in bytes, B = (KLEN-2HLEN-49+1+7)/8 + 4 = (KLEN-2HLEN-9)/8 (where "/" represents integer division rounding down).

The maximum length of message that can be encrypted using RSAEP-OAEP is defined by [RFC3447] in terms of the key length in bytes, which is (KLEN+7)/8. The maximum length is thus L = (KLEN+7-2HLEN-16)/8 = (KLEN-2HLEN-9)/8. Thus, the encoded version of K is always small enough to be encrypted under K_T.

Trademark Notice

"SSH" is a registered trademark in the United States.

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

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