Internet Engineering Task Force (IETF)
Request for Comments: 6239
Category: Informational
ISSN: 2070-1721

K. Igoe
National Security Agency

May 2011

Suite B Cryptographic Suites for Secure Shell (SSH)

Abstract

This document describes the architecture of a Suite B compliant implementation of the Secure Shell Transport Layer Protocol and the Secure Shell Authentication Protocol. Suite B Secure Shell makes use of the elliptic curve Diffie-Hellman (ECDH) key agreement, the elliptic curve digital signature algorithm (ECDSA), the Advanced Encryption Standard running in Galois/Counter Mode (AES-GCM), two members of the SHA-2 family of hashes (SHA-256 and SHA-384), and X.509 certificates.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

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). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6239.

Copyright Notice

Table of Contents

1. Introduction

This document describes the architecture of a Suite B compliant implementation of the Secure Shell Transport Layer Protocol and the Secure Shell Authentication Protocol. Suite B Secure Shell makes use of the elliptic curve Diffie-Hellman (ECDH) key agreement, the elliptic curve digital signature algorithm (ECDSA), the Advanced Encryption Standard running in Galois/Counter Mode (AES-GCM), two members of the SHA-2 family of hashes (SHA-256 and SHA-384), and X.509 certificates.

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 [RFC2119].

2. Suite B and Secure Shell

Several RFCs have documented how each of the Suite B components are to be integrated into Secure Shell (SSH):

```
kex algorithms
      ecdh-sha2-nistp256          [SSH-ECC]
      ecdh-sha2-nistp384          [SSH-ECC]

server host key algorithms
      x509v3-ecdsa-sha2-nistp256  [SSH-X509]
      x509v3-ecdsa-sha2-nistp384  [SSH-X509]

encryption algorithms (both client_to_server and server_to_client)
      AEAD_AES_128_GCM            [SSH-GCM]
      AEAD_AES_256_GCM            [SSH-GCM]

MAC algorithms (both client_to_server and server_to_client)
      AEAD_AES_128_GCM            [SSH-GCM]
      AEAD_AES_256_GCM            [SSH-GCM]
```

In Suite B, public key certificates used to verify signatures MUST be compliant with the Suite B Certificate Profile specified in RFC 5759 [SUITEBCERT].

The purpose of this document is to draw upon all of these documents to provide guidance for Suite B compliant implementations of Secure Shell (hereafter referred to as "SecSh-B"). Note that while SecSh-B MUST follow the guidance in this document, that requirement does not in and of itself imply that a given implementation of Secure Shell is suitable for use in protecting classified data. An implementation of SecSh-B must be validated by the appropriate authority before such usage is permitted.

The two elliptic curves used in Suite B appear in the literature under two different names. For the sake of clarity, we list both names below.

Curve	NIST name	SECG name	OID [SEC2]
P-256	nistp256	secp256r1	1.2.840.10045.3.1.7
P-384	nistp384	secp384r1	1.3.132.0.34

A description of these curves can be found in [NIST] or [SEC2].

For the sake of brevity, ECDSA-256 will be used to denote ECDSA on P-256 using SHA-256, and ECDSA-384 will be used to denote ECDSA on P-384 using SHA-384.

2.1. Minimum Levels of Security (minLOS)

Suite B provides for two levels of cryptographic security, namely a 128-bit minimum level of security (minLOS_128) and a 192-bit minimum level of security (minLOS_192). As we shall see below, the ECDSA-256/384 signature algorithms and corresponding X.509v3 certificates are treated somewhat differently than the non-signature primitives (kex algorithms, encryption algorithms, and Message Authentication Code (MAC) algorithms in Secure Shell parlance).

2.2. Digital Signatures and Certificates

SecSh-B uses ECDSA-256/384 for server authentication, user authentication, and in X.509 certificates. [SSH-X509] defines two methods, x509v3-ecdsa-sha2-nistp256 and x509v3-ecdsa-sha2-nistp384, that are to be used for server and user authentication. The following conditions must be met:

1) The server MUST share its public key with the host using an X.509v3 certificate as described in [SSH-X509]. This public key MUST be used to authenticate the server to the host using ECDSA-256 or ECDSA-384 as appropriate (see Section 3).

2) User authentication MUST begin with public key authentication using ECDSA-256/384 with X.509v3 certificates (see Section 4). Additional user authentication methods MAY be used, but only after the certificate-based ECDSA authentication has been successfully completed.

3) The X.509v3 certificates MUST use only the two Suite B digital signatures, ECDSA-256 and ECDSA-384.

4) ECDSA-256 MUST NOT be used to sign an ECDSA-384 public key.

5) ECDSA-384 MAY be used to sign an ECDSA-256 public key.

6) At minLOS_192, all SecSh-B implementations MUST be able to verify ECDSA-384 signatures.

7) At minLOS_128, all SecSh-B implementations MUST be able to verify ECDSA-256 signatures and SHOULD be able to verify ECDSA-384 signatures, unless it is absolutely certain that the implementation will never need to verify certificates originating from an authority that uses an ECDSA-384 signing key.

8) At minLOS_128, each SecSh-B server and each SecSh-B user MUST have either an ECDSA-256 signing key with a corresponding X.509v3 certificate, an ECDSA-384 signing key with a corresponding X.509v3 certificate, or both.

9) At minLOS_192, each SecSh-B server and each SecSh-B user MUST have an ECDSA-384 signing key with a corresponding X.509v3 certificate.

The selection of the signature algorithm to be used for server authentication is governed by the server_host_key_algorithms name-list in the SSH_MSG_KEXINIT packet (see Section 3.1). The key exchange and server authentication are performed by the SSH_MSG_KEXECDH_REPLY packets (see Section 4). User authentication is done via the SSH_MSG_USERAUTH_REQUEST messages (see Section 5).

2.3. Non-Signature Primitives

This section covers the constraints that the choice of minimum security level imposes upon the selection of a key agreement protocol (kex algorithm), encryption algorithm, and data integrity algorithm (MAC algorithm). We divide the non-signature algorithms into two families, as shown in Table 1.

Algorithm	Family 1	Family 2
kex	ecdh-sha2-nistp256	ecdh-sha2-nistp384
encryption	AEAD_AES_128_GCM	AEAD_AES_256_GCM
MAC	AEAD_AES_128_GCM	AEAD_AES_256_GCM

Table 1. Families of Non-Signature Algorithms in SecSh-B

At the 128-bit minimum level of security:

- o The non-signature algorithms MUST either come exclusively from Family 1 or exclusively from Family 2.
- o The selection of Family 1 versus Family 2 is independent of the choice of server host key algorithm.

At the 192-bit minimum level of security:

- o The non-signature algorithms MUST all come from Family 2.

Most of the constraints described in this section can be achieved by severely restricting the kex_algorithm, encryption_algorithm, and mac_algorithm name lists offered in the SSH_MSG_KEXINIT packet. See Section 3.1 for details.

3. Security Mechanism Negotiation and Initialization

As described in [SSH-Tran], the exchange of SSH_MSG_KEXINIT between the server and the client establishes which key agreement algorithm, MAC algorithm, host key algorithm (server authentication algorithm), and encryption algorithm are to be used. This section describes how the Suite B components are to be used in the Secure Shell algorithm negotiation, key agreement, server authentication, and user authentication.

Negotiation and initialization of a Suite B Secure Shell connection involves the following Secure Shell messages (where C->S denotes a message from the client to the server, and S->C denotes a server-to-client message):

SSH_MSG_KEXINIT	C->S	Contains lists of algorithms acceptable to the client.
SSH_MSG_KEXINIT	S->C	Contains lists of algorithms acceptable to the server.
SSH_MSG_KEXECDH_INIT	C->S	Contains the client's ephemeral elliptic curve Diffie-Hellman key.
SSH_MSG_KEXECDH_REPLY	S->C	Contains a certificate with the server's ECDSA public signature key, the server's ephemeral ECDH contribution, and an ECDSA digital signature of the newly formed exchange hash value.

```
SSH_MSG_USERAUTH_REQUEST  C->S  Contains the user's name, the
                                name of the service the user is
                                requesting, the name of the
                                authentication method the client
                                wishes to use, and method-specific
                                fields.
```

When not in the midst of processing a key exchange, either party may initiate a key re-exchange by sending an SSH_MSG_KEXINIT packet. All packets exchanged during the re-exchange are encrypted and authenticated using the current keys until the conclusion of the re-exchange, at which point an SSH_MSG_NEWKEYS initiates a change to the newly established keys. Otherwise, the re-exchange protocol is identical to the initial key exchange protocol. See Section 9 of [SSH-Tran].

3.1. Algorithm Negotiation: SSH_MSG_KEXINIT

The choice of all but the user authentication methods are determined by the exchange of SSH_MSG_KEXINIT between the client and the server. As described in [SSH-Tran], the SSH_MSG_KEXINIT packet has the following structure:

```
byte         SSH_MSG_KEXINIT
byte[16]     cookie (random bytes)
name-list    kex_algorithms
name-list    server_host_key_algorithms
name-list    encryption_algorithms_client_to_server
name-list    encryption_algorithms_server_to_client
name-list    mac_algorithms_client_to_server
name-list    mac_algorithms_server_to_client
name-list    compression_algorithms_client_to_server
name-list    compression_algorithms_server_to_client
name-list    languages_client_to_server
name-list    languages_server_to_client
boolean      first_kex_packet_follows
uint32       0 (reserved for future extension)
```

The SSH_MSG_KEXINIT name lists can be used to constrain the choice of non-signature and host key algorithms in accordance with the guidance given in Section 2. Table 2 lists three allowable name lists for the non-signature algorithms. One of these options MUST be used.

```
      Family 1 only (min_LOS 128):
         kex_algorithm name_list         := { ecdh_sha2_nistp256 }
         encryption_algorithm name_list  := { AEAD_AES_128_GCM   }
         mac_algorithm name_list         := { AEAD_AES_128_GCM   }

      Family 2 only (min_LOS 128 or 192):
         kex_algorithm name_list         := { ecdh_sha2_nistp384 }
         encryption_algorithm name_list  := { AEAD_AES_256_GCM   }
         mac_algorithm name_list         := { AEAD_AES_256_GCM   }

      Family 1 or Family 2 (min_LOS 128):
         kex_algorithm name_list         := { ecdh_sha2_nistp256,
                                              ecdh_sha2_nistp384 }
         encryption_algorithm name_list  := { AEAD_AES_128_GCM,
                                              AEAD_AES_256_GCM   }
         mac_algorithm name_list         := { AEAD_AES_128_GCM,
                                              AEAD_AES_256_GCM   }
```

Table 2. Allowed Non-Signature Algorithm Name Lists

Table 3 lists three allowable name lists for the server host key algorithms. One of these options MUST be used.

```
           ECDSA-256 only (min_LOS 128):
              server_host_key_algorithms name_list :=
                               { x509v3-ecdsa-sha2-nistp256 }

           ECDSA-384 only (min_LOS 128 or 192):
              server_host_key_algorithms name_list :=
                               { x509v3-ecdsa-sha2-nistp384 }

           ECDSA-256 or ECDSA-384 (min_LOS 128):
              server_host_key_algorithms name_list :=
                               { x509v3-ecdsa-sha2-nistp256,
                                 x509v3-ecdsa-sha2-nistp384 }
```

Table 3. Allowed Server Host Key Algorithm Name Lists

4. Key Exchange and Server Authentication

SecSh-B uses ECDH to establish a shared secret value between the client and the server. An X.509v3 certificate containing the server's public signing ECDSA key and an ECDSA signature on the exchange hash value derived from the newly established shared secret value are used to authenticate the server to the client.

4.1. SSH_MSG_KEXECDH_INIT

The key exchange to be used in Secure Shell is determined by the name lists exchanged in the SSH_MSG_KEXINIT packets. In Suite B, one of the following key agreement methods MUST be used to generate a shared secret value (SSV):

```
ecdh-sha2-nistp256      ephemeral-ephemeral elliptic curve
                        Diffie-Hellman on nistp256 with SHA-256

ecdh-sha2-nistp384      ephemeral-ephemeral elliptic curve
                        Diffie-Hellman on nistp384 with SHA-384
```

and the format of the SSH_MSG_KEXECDH_INIT message is:

```
byte      SSH_MSG_KEXDH_INIT

string    Q_C    // the client's ephemeral contribution to the
                 // ECDH exchange, encoded as an octet string
```

where the encoding of the elliptic curve point Q_C as an octet string is as specified in Section 2.3.3 of [SEC1].

4.2. SSH_MSG_KEXECDH_REPLY

The SSH_MSG_KEXECDH_REPLY contains the server's contribution to the ECDH exchange, the server's public signature key, and a signature of the exchange hash value formed from the newly established shared secret value. As stated in Section 3.1, in SecSh-B, the server host key algorithm MUST be either x509v3-ecdsa-sha2-nistp256 or x509v3-ecdsa-sha2-nistp384.

The format of the SSH_MSG_KEXECDH_REPLY is:

```
byte      SSH_MSG_KEXECDH_REPLY

string    K_S    // a string encoding an X.509v3 certificate
                 // containing the server's ECDSA public host key

string    Q_S    // the server's ephemeral contribution to the
                 // ECDH exchange, encoded as an octet string

string    Sig_S  // an octet string containing the server's
                 // signature of the newly established exchange
                 // hash value
```

Details on the structure and encoding of the X.509v3 certificate can be found in Section 2 of [SSH-X509]. The encoding of the elliptic curve point Q_C as an octet string is as specified in Section 2.3.3 of [SEC1], and the encoding of the ECDSA signature Sig_S as an octet string is as described in Section 3.1.2 of [SSH-ECC].

4.3. Key and Initialization Vector Derivation

As specified in [SSH-Tran], the encryption keys and initialization vectors needed by Secure Shell are derived directly from the SSV using the hash function specified by the key agreement algorithm (SHA-256 for ecdh-sha2-nistp256 and SHA-384 for ecdh-sha2-nistp384). The client-to-server channel and the server-to-client channel will have independent keys and initialization vectors. These keys will remain constant until a re-exchange results in the formation of a new SSV.

5. User Authentication

The Secure Shell Transport Layer Protocol authenticates the server to the host but does not authenticate the user (or the user's host) to the server. For this reason, condition (2) of Section 2.2 requires that all users of SecSh-B MUST be authenticated using ECDSA-256/384 signatures and X.509v3 certificates. [SSH-X509] provides two methods, x509v3-ecdsa-sha2-nistp256 and x509v3-ecdsa-sha2-nistp384, that MUST be used to achieve this goal. At minLOS 128, either one of these methods may be used, but at minLOS 192, x509v3-ecdsa-sha2-nistp384 MUST be used.

5.1. First SSH_MSG_USERAUTH_REQUEST Message

The user's public key is sent to the server using an SSH_MSG_USERAUTH_REQUEST message. When an x509v3-ecdsa-sha2-* user authentication method is being used, the structure of the SSH_MSG_USERAUTH_REQUEST message should be:

```
byte      SSH_MSG_USERAUTH_REQUEST

string    user_name      // in ISO-10646 UTF-8 encoding

string    service_name   // service name in US-ASCII

string    "publickey"

boolean   FALSE
```

```
      string    public_key_algorithm_name  // x509v3-ecdsa-sha2-nistp256
                                          // or x509v3-ecdsa-sha2-nistp384

      string    public_key_blob // X.509v3 certificate
```

Details on the structure and encoding of the X.509v3 certificate can be found in Section 2 of [SSH-X509].

5.2. Second SSH_MSG_USERAUTH_REQUEST Message

Once the server has responded to the request message with an SSH_MSG_USERAUTH_PK_OK message, the client uses a second SSH_MSG_USERAUTH_REQUEST message to perform the actual authentication:

```
      byte      SSH_MSG_USERAUTH_REQUEST

      string    user_name      // in ISO-10646 UTF-8 encoding

      string    service_name   // service name in US-ASCII

      string    "publickey"

      boolean   TRUE

      string    public_key_algorithm_name  // x509v3-ecdsa-sha2-nistp256
                                          // or x509v3-ecdsa-sha2-nistp384

      string    Sig_U
```

The signature field Sig_U is an ECDSA signature of the concatenation of several values, including the session identifier, user name, service name, public key algorithm name, and the user's public signing key. The user's public signing key MUST be the signing key conveyed in the X.509v3 certificate sent in the first SSH_MSG_USERAUTH_REQUEST message. The encoding of the ECDSA signature Sig_U as an octet string is as described in Section 3.1.2 of [SSH-ECC].

The server MUST respond with either SSH_MSG_USERAUTH_SUCCESS (if no more authentications are needed) or SSH_MSG_USERAUTH_FAILURE (if the request failed, or more authentications are needed).

6. Confidentiality and Data Integrity of SSH Binary Packets

Secure Shell transfers data between the client and the server using its own binary packet structure. The SSH binary packet structure is independent of any packet structure on the underlying data channel. The contents of each binary packet and portions of the header are encrypted, and each packet is authenticated with its own message authentication code. AES GCM will both encrypt the packet and form a 16-octet authentication tag to ensure data integrity.

6.1. Galois/Counter Mode

[SSH-GCM] describes how AES Galois/Counter Mode is to be used in Secure Shell. Suite B SSH implementations MUST support AEAD_AES_GCM_128 and SHOULD support AEAD_AES_GCM_256 to both provide confidentiality and ensure data integrity. No other confidentiality or data integrity algorithms are permitted.

These algorithms rely on two counters:

Invocation Counter: A 64-bit integer, incremented after each call to AES-GCM to process an SSH binary packet. The initial value of the invocation counter is determined by the SSH initialization vector.

Block Counter: A 32-bit integer, set to one at the start of each new SSH binary packet and incremented as each 16-octet block of data is processed.

Ensuring that these counters are properly implemented is crucial to the security of the system. The reader is referred to [SSH-GCM] for details on the format, initialization, and usage of these counters and their relationship to the initialization vector and the SSV.

6.2. Data Integrity

The reader is reminded that, as specified in [SSH-GCM], Suite B requires that all 16 octets of the authentication tag MUST be used as the SSH data integrity value of the SSH binary packet.

7. Rekeying

Secure Shell allows either the server or client to request that the Secure Shell connection be rekeyed. Suite B places no constraints on how frequently this is to be done, but it does require that the cipher suite being employed MUST NOT be changed when a rekey occurs.

8. Security Considerations

When using ecdh_sha2_nistp256, each exponent used in the key exchange must have 256 bits of entropy. Similarly, when using ecdh_sha2_nistp384, each exponent used in the key exchange must have 384 bits of entropy. The security considerations of [SSH-Arch] apply.

9. References

9.1. Normative References

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

[SUITEBCERT] Solinas, J. and L. Zieglar, "Suite B Certificate and Certificate Revocation List (CRL) Profile", RFC 5759, January 2010.

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

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

[SSH-ECC] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, December 2009.

[SSH-GCM] Igoe, K. and J. Solinas, "AES Galois Counter Mode for the Secure Shell Transport Layer Protocol", RFC 5647, August 2009.

[SSH-X509] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, March 2011.

9.2. Informative References

[NIST] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-3.

[SEC1] Standards for Efficient Cryptography Group, "Elliptic Curve Cryptography", SEC 1 v2.0, May 2009, <http://www.secg.org/download/aid-780/sec1-v2.pdf>.

[SEC2] Standards for Efficient Cryptography Group, "Recommended Elliptic Curve Domain Parameters", SEC 2 v1.0, September 2000. <http://www.secg.org/download/aid-386/sec2_final.pdf>.

Author's Address

Kevin M. Igoe
NSA/CSS Commercial Solutions Center
National Security Agency

EMail: kmigoe@nsa.gov