Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer

Abstract

   This document describes algorithms based on Elliptic Curve
   Cryptography (ECC) for use within the Secure Shell (SSH) transport
   protocol. In particular, it specifies Elliptic Curve Diffie-Hellman
   (ECDH) key agreement, Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key
   agreement, and Elliptic Curve Digital Signature Algorithm (ECDSA) for
   use in the SSH Transport Layer protocol.

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

outside the IETF Standards Process, and derivative works of it may
not be created outside the IETF Standards Process, except to format
it for publication as an RFC or to translate it into languages other
than English.

Table of Contents

1. Introduction

   This document adds the following elliptic curve cryptography
   algorithms to the Secure Shell arsenal: Elliptic Curve Diffie-Hellman
   (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA), as
   well as utilizing the SHA2 family of secure hash algorithms.
   Additionally, support is provided for Elliptic Curve Menezes-Qu-
   Vanstone (ECMQV).

   Due to its small key sizes and its inclusion in the National Security
   Agency's Suite B, Elliptic Curve Cryptography (ECC) is becoming a
   widely utilized and attractive public-key cryptosystem.

   Compared to cryptosystems such as RSA, the Digital Signature
   Algorithm (DSA), and Diffie-Hellman (DH) key exchange, ECC variations
   on these schemes offer equivalent security with smaller key sizes.
   This is illustrated in the following table, based on Section 5.6.1 of
   NIST 800-57 [NIST-800-57], which gives approximate comparable key
   sizes for symmetric- and asymmetric-key cryptosystems based on the
   best known algorithms for attacking them. L is the field size and N
   is the sub-field size.

Symmetric	Discrete Log (e.g., DSA, DH)	RSA	ECC
80	L = 1024, N = 160	1024	160-223
112	L = 2048, N = 256	2048	224-255
128	L = 3072, N = 256	3072	256-383
192	L = 7680, N = 384	7680	384-511
256	L = 15360, N = 512	15360	512+

   Implementation of this specification requires familiarity with both
   SSH [RFC4251] [RFC4253] [RFC4250] and ECC [SEC1] (additional
   information on ECC available in [HMV04], [ANSI-X9.62], and
   [ANSI-X9.63]).

   This document is concerned with SSH implementation details;
   specification of the underlying cryptographic algorithms is left to
   other standards documents.

2. Notation

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].

The data types boolean, byte, uint32, uint64, string, and mpint are to be interpreted in this document as described in [RFC4251].

The size of a set of elliptic curve domain parameters on a prime curve is defined as the number of bits in the binary representation of the field order, commonly denoted by p. Size on a characteristic-2 curve is defined as the number of bits in the binary representation of the field, commonly denoted by m. A set of elliptic curve domain parameters defines a group of order n generated by a base point P.

3. SSH ECC Public Key Algorithm

The SSH ECC public key algorithm is defined by its key format, corresponding signature algorithm ECDSA, signature encoding, and algorithm identifiers.

This section defines the family of "ecdsa-sha2-*" public key formats and corresponding signature formats. Every compliant SSH ECC implementation MUST implement this public key format.

3.1. Key Format

The "ecdsa-sha2-*" key formats all have the following encoding:

    string   "ecdsa-sha2-[identifier]"
    byte[n] ecc_key_blob

The ecc_key_blob value has the following specific encoding:

    string   [identifier]
    string   Q

The string [identifier] is the identifier of the elliptic curve domain parameters. The format of this string is specified in Section 6.1. Information on the REQUIRED and RECOMMENDED sets of elliptic curve domain parameters for use with this algorithm can be found in Section 10.

Q is the public key encoded from an elliptic curve point into an octet string as defined in Section 2.3.3 of [SEC1]; point compression MAY be used.

The algorithm for ECC key generation can be found in Section 3.2 of
[SEC1]. Given some elliptic curve domain parameters, an ECC key pair
can be generated containing a private key (an integer d), and a
public key (an elliptic curve point Q).

3.1.1. Signature Algorithm

Signing and verifying is done using the Elliptic Curve Digital
Signature Algorithm (ECDSA). ECDSA is specified in [SEC1]. The
message hashing algorithm MUST be from the SHA2 family of hash
functions [FIPS-180-3] and is chosen according to the curve size as
specified in Section 6.2.1.

3.1.2. Signature Encoding

Signatures are encoded as follows:

    string   "ecdsa-sha2-[identifier]"
    string   ecdsa_signature_blob

The string [identifier] is the identifier of the elliptic curve
domain parameters. The format of this string is specified in
Section 6.1. Information on the REQUIRED and RECOMMENDED sets of
elliptic curve domain parameters for use with this algorithm can be
found in Section 10.

The ecdsa_signature_blob value has the following specific encoding:

    mpint   r
    mpint   s

The integers r and s are the output of the ECDSA algorithm.

The width of the integer fields is determined by the curve being
used. Note that the integers r and s are integers modulo the order
of the cryptographic subgroup, which may be larger than the size of
the finite field.

4. ECDH Key Exchange

The Elliptic Curve Diffie-Hellman (ECDH) key exchange method
generates a shared secret from an ephemeral local elliptic curve
private key and ephemeral remote elliptic curve public key. This key
exchange method provides explicit server authentication as defined in
[RFC4253] using a signature on the exchange hash. Every compliant
SSH ECC implementation MUST implement ECDH key exchange.

The primitive used for shared key generation is ECDH with cofactor
multiplication, the full specification of which can be found in
Section 3.3.2 of [SEC1]. The algorithm for key pair generation can
be found in Section 3.2.1 of [SEC1].

The family of key exchange method names defined for use with this key
exchange can be found in Section 6.3. Algorithm negotiation chooses
the public key algorithm to be used for signing and the method name
of the key exchange. The method name of the key exchange chosen
determines the elliptic curve domain parameters and hash function to
be used in the remainder of this section.

Information on the REQUIRED and RECOMMENDED elliptic curve domain
parameters for use with this method can be found in Section 10.

All elliptic curve public keys MUST be validated after they are
received. An example of a validation algorithm can be found in
Section 3.2.2 of [SEC1]. If a key fails validation, the key exchange
MUST fail.

The elliptic curve public keys (points) that must be transmitted are
encoded into octet strings before they are transmitted. The
transformation between elliptic curve points and octet strings is
specified in Sections 2.3.3 and 2.3.4 of [SEC1]; point compression
MAY be used. The output of shared key generation is a field element
xp. The SSH framework requires that the shared key be an integer.
The conversion between a field element and an integer is specified in
Section 2.3.9 of [SEC1].

Specification of the message numbers SSH_MSG_KEX_ECDH_INIT and
SSH_MSG_KEX_ECDH_REPLY is found in Section 7.

The following is an overview of the key exchange process:

```
Client                                                 Server
------                                                 ------
Generate ephemeral key pair.
SSH_MSG_KEX_ECDH_INIT  -------------->

                                      Verify received key is valid.
                                       Generate ephemeral key pair.
                                            Compute shared secret.
                                     Generate and sign exchange hash.
                           <------------- SSH_MSG_KEX_ECDH_REPLY

Verify received key is valid.
*Verify host key belongs to server.
Compute shared secret.
Generate exchange hash.
Verify server's signature.
```

   * It is RECOMMENDED that the client verify that the host key sent
      is the server's host key (for example, using a local database).
      The client MAY accept the host key without verification, but
      doing so will render the protocol insecure against active
      attacks; see the discussion in Section 4.1 of [RFC4251].

   This is implemented using the following messages.

   The client sends:

```
byte        SSH_MSG_KEX_ECDH_INIT
string      Q_C, client's ephemeral public key octet string
```

   The server responds with:

```
byte        SSH_MSG_KEX_ECDH_REPLY
string      K_S, server's public host key
string      Q_S, server's ephemeral public key octet string
string      the signature on the exchange hash
```

The exchange hash H is computed as the hash of the concatenation of
the following.

```
   string    V_C, client's identification string (CR and LF excluded)
   string    V_S, server's identification string (CR and LF excluded)
   string    I_C, payload of the client's SSH_MSG_KEXINIT
   string    I_S, payload of the server's SSH_MSG_KEXINIT
   string    K_S, server's public host key
   string    Q_C, client's ephemeral public key octet string
   string    Q_S, server's ephemeral public key octet string
   mpint     K,   shared secret
```

5. ECMQV Key Exchange

The Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key exchange algorithm
generates a shared secret from two local elliptic curve key pairs and
two remote public keys. This key exchange method provides implicit
server authentication as defined in [RFC4253]. The ECMQV key
exchange method is OPTIONAL.

The key exchange method name defined for use with this key exchange
is "ecmqv-sha2". This method name gives a hashing algorithm that is
to be used for the Hashed Message Authentication Code (HMAC) below.
Future RFCs may define new method names specifying new hash
algorithms for use with ECMQV. More information about the method
name and HMAC can be found in Section 6.4.

In general, the ECMQV key exchange is performed using the ephemeral
and long-term key pair of both the client and server, which is a
total of 4 keys. Within the framework of SSH, the client does not
have a long-term key pair that needs to be authenticated. Therefore,
we generate an ephemeral key and use that as both the clients keys.
This is more efficient than using two different ephemeral keys, and
it does not adversely affect security (it is analogous to the one-
pass protocol in Section 6.1 of [LMQSV98]).

A full description of the ECMQV primitive can be found in Section 3.4
of [SEC1]. The algorithm for key pair generation can be found in
Section 3.2.1 of [SEC1].

During algorithm negotiation with the SSH_MSG_KEXINIT messages, the
ECMQV key exchange method can only be chosen if a public key
algorithm supporting ECC host keys can also be chosen. This is due
to the use of implicit server authentication in this key exchange
method. This case is handled the same way that key exchange methods
requiring encryption/signature capable public key algorithms are

handled in Section 7.1 of [RFC4253]. If ECMQV key exchange is
chosen, then the public key algorithm supporting ECC host keys MUST
also be chosen.

ECMQV requires that all the keys used to generate a shared secret are
generated over the same elliptic curve domain parameters. Since the
host key is used in the generation of the shared secret, allowing for
implicit server authentication, the domain parameters associated with
the host key are used throughout this section.

All elliptic curve public keys MUST be validated after they are
received. An example of a validation algorithm can be found in
Section 3.2.2 of [SEC1]. If a key fails validation, the key exchange
MUST fail.

The elliptic curve ephemeral public keys (points) that must be
transmitted are encoded into octet strings before they are
transmitted. The transformation between elliptic curve points and
octet strings is specified in Sections 2.3.3 and 2.3.4 of [SEC1];
point compression MAY be used. The output of shared key generation
is a field element x_p. The SSH framework requires that the shared
key be an integer. The conversion between a field element and an
integer is specified in Section 2.3.9 of [SEC1].

The following is an overview of the key exchange process:

```
    Client                                              Server
    ------                                              ------
    Generate ephemeral key pair.
    SSH_MSG_KEX_ECMQV_INIT ------------->

                                        Verify received key is valid.
                                         Generate ephemeral key pair.
                                               Compute shared secret.
                                        Generate exchange hash and compute
                                        HMAC over it using the shared secret.
                                 <------------- SSH_MSG_KEX_ECMQV_REPLY

    Verify received keys are valid.
    *Verify host key belongs to server.
    Compute shared secret.
    Verify HMAC.
```

    * It is RECOMMENDED that the client verify that the host key sent
       is the server's host key (for example, using a local database).
       The client MAY accept the host key without verification, but
       doing so will render the protocol insecure against active
       attacks.

The specification of the message numbers SSH_MSG_ECMQV_INIT and
SSH_MSG_ECMQV_REPLY can be found in Section 7.

This key exchange algorithm is implemented with the following
messages.

The client sends:

```
byte      SSH_MSG_ECMQV_INIT
string    Q_C, client's ephemeral public key octet string
```

The server sends:

```
byte      SSH_MSG_ECMQV_REPLY
string    K_S, server's public host key
string    Q_S, server's ephemeral public key octet string
string    HMAC tag computed on H using the shared secret
```

The hash H is formed by applying the algorithm HASH on a
concatenation of the following:

```
string    V_C, client's identification string (CR and LF excluded)
string    V_S, server's identification string (CR and LF excluded)
string    I_C, payload of the client's SSH_MSG_KEXINIT
string    I_S, payload of the server's SSH_MSG_KEXINIT
string    K_S, server's public host key
string    Q_C, client's ephemeral public key octet string
string    Q_S, server's ephemeral public key octet string
mpint     K,   shared secret
```

6. Method Names

This document defines a new family of key exchange method names, a
new key exchange method name, and a new family of public key
algorithm names in the SSH name registry.

6.1. Elliptic Curve Domain Parameter Identifiers

This section specifies identifiers encoding named elliptic curve
domain parameters. These identifiers are used in this document to
identify the curve used in the SSH ECC public key format, the ECDSA
signature blob, and the ECDH method name.

For the REQUIRED elliptic curves nistp256, nistp384, and nistp521,
the elliptic curve domain parameter identifiers are the strings
"nistp256", "nistp384", and "nistp521".

For all other elliptic curves, including all other NIST curves and
all other RECOMMENDED curves, the elliptic curve domain parameter
identifier is the ASCII period-separated decimal representation of
the Abstract Syntax Notation One (ASN.1) [ASN1] Object Identifier
(OID) of the named curve domain parameters that are associated with
the server's ECC host keys. This identifier is defined provided that
the concatenation of the public key format identifier and the
elliptic curve domain parameter identifier (or the method name and
the elliptic curve domain parameter identifier) does not exceed the
maximum specified by the SSH protocol architecture [RFC4251], namely
64 characters; otherwise, the identifier for that curve is undefined,
and the curve is not supported by this specification.

A list of the REQUIRED and RECOMMENDED curves and their OIDs can be
found in Section 10.

Note that implementations MUST use the string identifiers for the
three REQUIRED NIST curves, even when an OID exists for that curve.

6.2. ECC Public Key Algorithm (ecdsa-sha2-*)

The SSH ECC public key algorithm is specified by a family of public
key format identifiers. Each identifier is the concatenation of the
string "ecdsa-sha2-" with the elliptic curve domain parameter
identifier as defined in Section 6.1. A list of the required and
recommended curves and their OIDs can be found in Section 10.

For example, the method name for ECDH key exchange with ephemeral
keys generated on the nistp256 curve is "ecdsa-sha2-nistp256".

6.2.1. Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is specified
for use with the SSH ECC public key algorithm.

The hashing algorithm defined by this family of method names is the
SHA2 family of hashing algorithms [FIPS-180-3]. The algorithm from
the SHA2 family that will be used is chosen based on the size of the
named curve specified in the public key:

Curve Size	Hash Algorithm
b <= 256	SHA-256
256 < b <= 384	SHA-384
384 < b	SHA-512

6.3. ECDH Key Exchange Method Names (ecdh-sha2-*)

The Elliptic Curve Diffie-Hellman (ECDH) key exchange is defined by a family of method names. Each method name is the concatenation of the string "ecdh-sha2-" with the elliptic curve domain parameter identifier as defined in Section 6.1. A list of the required and recommended curves and their OIDs can be found in Section 10.

For example, the method name for ECDH key exchange with ephemeral keys generated on the sect409k1 curve is "ecdh-sha2-1.3.132.0.36".

The hashing algorithm defined by this family of method names is the SHA2 family of hashing algorithms [FIPS-180-3]. The hashing algorithm is defined in the method name to allow room for other algorithms to be defined in future documents. The algorithm from the SHA2 family that will be used is chosen based on the size of the named curve specified in the method name according to the table in Section 6.2.1.

The concatenation of any so encoded ASN.1 OID specifying a set of elliptic curve domain parameters with "ecdh-sha2-" is implicitly registered under this specification.

6.4. ECMQV Key Exchange and Verification Method Name (ecmqv-sha2)

The Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key exchange is defined by the method name "ecmqv-sha2". Unlike the ECDH key exchange method, ECMQV relies on a public key algorithm that uses ECC keys: it does not need a family of method names because the curve information can be gained from the public key algorithm.

The hashing and message authentication code algorithms are defined by the method name to allow room for other algorithms to be defined for use with ECMQV in future documents.

The hashing algorithm defined by this method name is the SHA2 family
of hashing algorithms [FIPS-180-3]. The algorithm from the SHA2
family that will be used is chosen based on the size of the named
curve specified for use with ECMQV by the chosen public key algorithm
according to the table in Section 6.2.1.

The keyed-hash message authentication code that is used to identify
the server and verify communications is based on the hash chosen
above. The information on implementing the HMAC based on the chosen
hash algorithm can be found in [RFC2104].

7. Key Exchange Messages

The message numbers 30-49 are key-exchange-specific and in a private
namespace defined in [RFC4250] that may be redefined by any key
exchange method [RFC4253] without requiring an IANA registration
process.

The following message numbers have been defined in this document:

7.1. ECDH Message Numbers

```
#define SSH_MSG_KEX_ECDH_INIT              30
#define SSH_MSG_KEX_ECDH_REPLY             31
```

7.2. ECMQV Message Numbers

```
#define SSH_MSG_ECMQV_INIT                 30
#define SSH_MSG_ECMQV_REPLY                31
```

8. Manageability Considerations

As this document only provides new public key algorithms and key
exchange methods within the existing Secure Shell protocol
architecture, there are few manageability considerations beyond those
that apply for existing Secure Shell implementations. Additional
manageability considerations are listed below.

8.1. Control of Function through Configuration and Policy

Section 10 specifies REQUIRED and RECOMMENDED elliptic curve domain
parameters to be used with the public key algorithms and key exchange
methods defined in this document. Implementers SHOULD allow system
administrators to disable some curves, including REQUIRED or
RECOMMENDED curves, to meet local security policy.

8.2. Impact on Network Operation

As this document provides new functionality within the Secure Shell
protocol architecture, the only impact on network operations is the
impact on existing Secure Shell implementations. The Secure Shell
protocol provides negotiation mechanisms for public key algorithms
and key exchange methods: any implementations that do not recognize
the algorithms and methods defined in this document will ignore them
in the negotiation and use the next mutually supported algorithm or
method, causing no negative impact on backward compatibility.

The use of elliptic curve cryptography should not place a significant
computational burden on an implementing server. In fact, due to its
smaller key sizes, elliptic curve cryptography can be implemented
more efficiently for the same security level than RSA, finite field
Diffie-Hellman, or DSA.

9. Security Considerations

This document provides new public key algorithms and new key
agreement methods for the Secure Shell protocol. For the most part,
the security considerations involved in using the Secure Shell
protocol apply. Additionally, implementers should be aware of
security considerations specific to elliptic curve cryptography.

For all three classes of functionality added by this document (the
public key algorithms involving ECDSA, key exchange involving ECDH,
and authenticated key exchange involving ECMQV), the current best
known technique for breaking the cryptosystems is by solving the
elliptic curve discrete logarithm problem (ECDLP).

The difficulty of breaking the ECDLP depends on the size and quality
of the elliptic curve parameters. Certain types of curves can be
more susceptible to known attacks than others. For example, curves
over finite fields GF(2^m), where m is composite, may be susceptible
to an attack based on the Weil descent. All of the RECOMMENDED
curves in Section 10 do not have this problem. System administrators
should be cautious when enabling curves other than the ones specified
in Section 10 and should make a more detailed investigation into the
security of the curve in question.

It is believed (see, for example, Section B.2.1 of [SEC1]) that when
curve parameters are generated at random, the curves will be
resistant to special attacks, and must rely on the most general
attacks. The REQUIRED curves in Section 10 were all generated
verifiably pseudorandomly. The runtime of general attacks depends on
the algorithm used. At present, the best known algorithm is the
Pollard-rho method. (Shor's algorithm for quantum computers can

solve the ECDLP in polynomial time, but at present large-scale
quantum computers have not been constructed and significant
experimental physics and engineering work needs to be done before
large-scale quantum computers can be constructed. There is no solid
estimate as to when this may occur, but it is widely believed to be
at least 20 years from the present.)

Based on projections of computation power, it is possible to estimate
the running time of the best known attacks based on the size of the
finite field. The table in Section 1 gives an estimate of the
equivalence between elliptic curve field size and symmetric key size.
Roughly speaking, an N-bit elliptic curve offers the same security as
an N/2-bit symmetric cipher, so a 256-bit elliptic curve (such as the
REQUIRED nistp256 curve) is suitable for use with 128-bit AES, for
example.

Many estimates consider that 2^{80}-2^{90} operations are beyond
feasible, so that would suggest using elliptic curves of at least
160-180 bits. The REQUIRED curves in this document are 256-, 384-,
and 521-bit curves; implementations SHOULD NOT use curves smaller
than 160 bits.

A detailed discussion on the security considerations of elliptic
curve domain parameters and the ECDH, ECDSA, and ECMQV algorithms can
be found in Appendix B of [SEC1].

Additionally, the key exchange methods defined in this document rely
on the SHA2 family of hash functions, defined in [FIPS-180-3]. The
appropriate security considerations of that document apply. Although
some weaknesses have been discovered in the predecessor, SHA-1, no
weaknesses in the SHA2 family are known at present. The SHA2 family
consists of four variants -- SHA-224, SHA-256, SHA-384, and SHA-521
-- named after their digest lengths. In the absence of special
purpose attacks exploiting the specific structure of the hash
function, the difficulty of finding collisions, preimages, and second
preimages for the hash function is related to the digest length.
This document specifies in Section 6.2.1 which SHA2 variant should be
used with which elliptic curve size based on this guidance.

Since ECDH and ECMQV allow for elliptic curves of arbitrary sizes and
thus arbitrary security strength, it is important that the size of
elliptic curve be chosen to match the security strength of other
elements of the SSH handshake. In particular, host key sizes,
hashing algorithms and bulk encryption algorithms must be chosen
appropriately. Information regarding estimated equivalence of key
sizes is available in [NIST-800-57]; the discussion in [RFC3766] is
also relevant. We note in particular that when ECDSA is used as the

signature algorithm and ECDH is used as the key exchange method, if
curves of different sizes are used, then it is possible that
different hash functions from the SHA2 family could be used.

The REQUIRED and RECOMMENDED curves in this document are at present
believed to offer security at the levels indicated in this section
and as specified with the table in Section 1.

System administrators and implementers should take careful
consideration of the security issues when enabling curves other than
the REQUIRED or RECOMMENDED curves in this document. Not all
elliptic curves are secure, even if they are over a large field.

Implementers SHOULD ensure that any ephemeral private keys or random
values -- including the value k used in ECDSA signature generation
and the ephemeral private key values in ECDH and ECMQV -- are
generated from a random number generator or a properly seeded
pseudorandom number generator, are protected from leakage, are not
reused outside of the context of the protocol in this document, and
are erased from memory when no longer needed.

10. Named Elliptic Curve Domain Parameters

Implementations MAY support any ASN.1 object identifier (OID) in the
ASN.1 object tree that defines a set of elliptic curve domain
parameters [ASN1].

10.1. Required Curves

Every SSH ECC implementation MUST support the named curves below.
These curves are defined in [SEC2]; the NIST curves were originally
defined in [NIST-CURVES]. These curves SHOULD always be enabled
unless specifically disabled by local security policy.

NIST*	SEC	OID
nistp256	secp256r1	1.2.840.10045.3.1.7
nistp384	secp384r1	1.3.132.0.34
nistp521	secp521r1	1.3.132.0.35

  * For these three REQUIRED curves, the elliptic curve domain
     parameter identifier is the string in the first column of the
     table, the NIST name of the curve. (See Section 6.1.)

10.2. Recommended Curves

It is RECOMMENDED that SSH ECC implementations also support the
following curves. These curves are defined in [SEC2].

NIST	SEC	OID*
nistk163	sect163k1	1.3.132.0.1
nistp192	secp192r1	1.2.840.10045.3.1.1
nistp224	secp224r1	1.3.132.0.33
nistk233	sect233k1	1.3.132.0.26
nistb233	sect233r1	1.3.132.0.27
nistk283	sect283k1	1.3.132.0.16
nistk409	sect409k1	1.3.132.0.36
nistb409	sect409r1	1.3.132.0.37
nistt571	sect571k1	1.3.132.0.38

   * For these RECOMMENDED curves, the elliptic curve domain
      parameter identifier is the string in the third column of the
      table, the ASCII representation of the OID of the curve. (See
      Section 6.1.)

11. IANA Considerations

Consistent with Section 8 of [RFC4251] and Section 4.6 of [RFC4250],
this document makes the following registrations:

In the Public Key Algorithm Names registry: The family of SSH public
key algorithm names beginning with "ecdsa-sha2-" and not containing
the at-sign ('@'), to name the public key algorithms defined in
Section 3.

In the Key Exchange Method Names registry: The family of SSH key
exchange method names beginning with "ecdh-sha2-" and not containing
the at-sign ('@'), to name the key exchange methods defined in
Section 4.

In the Key Exchange Method Names registry: The SSH key exchange
method name "ecmqv-sha2" to name the key exchange method defined in
Section 5.

This document creates no new registries.

12. References

12.1. Normative References

[ASN1]   International Telecommunications Union, "Abstract
                Syntax Notation One (ASN.1): Specification of basic
                notation", X.680, July 2002.

[FIPS-180-3]   National Institute of Standards and Technology,
                "Secure Hash Standard", FIPS 180-3, October 2008.

[RFC2104]   Krawczyk, H., Bellare, M., and R. Canetti, "HMAC:
                Keyed-Hashing for Message Authentication", RFC 2104,
                February 1997.

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

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

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

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

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

[SEC1]   Standards for Efficient Cryptography Group, "Elliptic
                Curve Cryptography", SEC 1, 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,
                September 2000,
                <http://www.secg.org/download/aid-386/sec2_final.pdf>.

12.2. Informative References

[ANSI-X9.62]   American National Standards Institute, "Public Key
                Cryptography For The Financial Services Industry: The
                Elliptic Curve Digital Signature Algorithm (ECDSA)",
                ANSI X9.62, 1998.

[ANSI-X9.63]   American National Standards Institute, "Public Key
                Cryptography For The Financial Services Industry: Key
                Agreement and Key Transport Using Elliptic Curve
                Cryptography", ANSI X9.63, January 1999.

[HMV04]   Hankerson, D., Menezes, A., and S. Vanstone, "Guide to
                Elliptic Curve Cryptography", Springer ISBN
                038795273X, 2004.

[LMQSV98]   Law, L., Menezes, A., Qu, M., Solinas, J., and S.
                Vanstone, "An Efficient Protocol for Authenticated Key
                Agreement", University of Waterloo Technical Report
                CORR 98-05, August 1998, <http://
                www.cacr.math.uwaterloo.ca/techreports/1998/
                corr98-05.pdf>.

[NIST-800-57]   National Institute of Standards and Technology,
                "Recommendation for Key Management - Part 1: General
                (Revised)", NIST Special Publication 800-57,
                March 2007.

[NIST-CURVES]   National Institute of Standards and Technology,
                "Recommended Elliptic Curves for Federal Government
                Use", July 1999.

Appendix A. Acknowledgements

   The authors acknowledge helpful comments from James Blaisdell, David
   Harrington, Alfred Hoenes, Russ Housley, Jeffrey Hutzelman, Kevin
   Igoe, Rob Lambert, Jan Pechanek, Tim Polk, Sean Turner, Nicolas
   Williams, and members of the ietf-ssh@netbsd.org mailing list.

Authors' Addresses

   Douglas Stebila
   Queensland University of Technology
   Information Security Institute
   Level 7, 126 Margaret St
   Brisbane, Queensland 4000
   Australia

   EMail: douglas@stebila.ca

   Jon Green
   Queen's University
   Parallel Processing Research Laboratory
   Department of Electrical and Computer Engineering
   Room 614, Walter Light Hall
   Kingston, Ontario K7L 3N6
   Canada

   EMail: jonathan.green@queensu.ca