Secure Shell Working Group
Internet-Draft
Expires: October 30, 2004

D. Stebila
Sun Labs
May 1, 2004

Elliptic-Curve Diffie-Hellman Key Exchange for the SSH Transport Level Protocol
draft-stebila-secsh-ecdh-01

Abstract

This document describes new key exchange algorithms based on Elliptic Curve Cryptography (ECC) for the Secure Shell (SSH) protocol. In particular, it specifies the use of Elliptic Curve Diffie-Hellman (ECDH) key agreement and ECDH with cofactor multiplication key agreement in the SSH Transport Layer protocol.

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 [1].

Table of Contents

1. Introduction

Elliptic Curve Cryptography (ECC) is emerging as an attractive public-key cryptosystem for mobile/wireless environments. Compared to currently prevalent cryptosystems such as RSA, DSA, and DH, ECC offers equivalent security with smaller key sizes. This is illustrated in the following table, based on [2], which gives approximate comparable key sizes for symmetric- and asymmetric-key cryptosystems based on the best known algorithms for attacking them.

Symmetric	ECC	DH/DSA/RSA
80	163	1024
128	283	3072
192	409	7680
256	571	15360

Figure 1: Comparable key sizes (in bits)

Smaller key sizes result in power, bandwidth, and computational savings that make ECC especially attractive for constrained environments.

This document describes additions to SSH to support the use of the Elliptic Curve Diffie-Hellman (ECDH) key agreement scheme to establish a shared key, either with or without cofactor multiplication.

Implementation of this specification requires familiarity with both SSH [3][4] and ECC [5][6][7].

2. ECDH Parameter and Key Exchange

2.1 Description

The first stage of the key exchange mechanism is the exchange of elliptic curve domain parameters. These parameters define a finite field (either GF(p), a field of integers modulo a large prime p, or GF(2^m), the field of binary polynomials of degree m or less), an elliptic curve over that finite field, a point on that curve, and the order and cofactor of the subgroup generated by scalar-point multiplication by that point.

Random elliptic curve domain parameters can be generated upon each request. However, this is costly and may result in the generation of parameters the security of which is untested. As a result, there are a number of so-called named elliptic curve domain parameters defined by various standards organizations [8][9][10][11].

In the following: C is the client, S is the server; curves is a list of supported named curves and generic identifiers; min is the minimal size in bits of acceptable generic elliptic curve domain parameters, or 0; pref is the preferred size in bits of acceptable generic elliptic curve domain parameters, or 0; max is the maximal size in bits of acceptable generic elliptic curve domain parameters, or 0; curve is a named curve or generic identifier; prime is a prime defining a prime field; irr is an irreducible polynomial defining a binary field; a is the curve parameter a; b is the curve parameter b; x is the x-coordinate of the generator; y is the y-coordinate of the generator; order is the order of the generator; cofactor is the cofactor of the generator; seed is the the binary string used as a seed for random generation of the elliptic curve domain parameters, or ""; c_x is the x-coordinate of the client's public key; c_y is the y-coordinate of the client's public key; s_x is the x-coordinate of the server's public key; s_y is the y-coordinate of the server's public key; and K is the shared secret.

1. C sends "curves", a list of preferred named elliptic curve domain parameters and, optionally, a range of bit sizes "min || pref || max" over which generic elliptic curve domain parameters are acceptable.

2. S selects the first curve in the list of preferred curves that it supports. If S selects a named curve, S returns the name as "curve". If S selects a set of generic elliptic curve domain parameters, then S creates or selects a set of elliptic curve domain parameters in the requested range, if S supports a curve in that range; S sends the elliptic curve domain parameters to C: either "prime || a || b || x || y || order || cofactor || seed"

for a prime curve or "irr || a || b || x || y || order || cofactor || seed" for a binary curve.

3. If S sends a set of generic elliptic curve domain parameters, then C MAY verify that the curve was generated at random, using for example the verification algorithms defined in section A.16.8 of [6]. If C decides to accept the parameters, then C generates an elliptic curve public-key / private-key pair using the selected elliptic curve domain parameters and sends the public-key value "c_x || c_y" to S.

4. S MAY validate C's public-key value using for example algorithm A.16.10 of [6]. S generates an elliptic curve public-key / private-key pair using the selected elliptic curve domain parameters; the public-key value is "s_x || s_y". S uses C's public-key value to compute the shared secret K using the key derivation function KDF. S computes the hash H of all values transmitted in the key exchange concatenated with K, and the signature s on H using its private host key. S sends "K_S || s_x || s_y || s", where K_S is S's public host key.

5. C MAY validate S's public-key value using for example algorithm A.16.10 of [6]. C verifies that K_S really is the host key for S (e.g., using certificates or a local database). C is also allowed to accept the key without verification; however, doing so will render the protocol insecure against active attacks (but may be desirable for practical reasons in the short term in many environments). C uses S's public-key value to compute the shared secret using the key derivation function KDF and verifies the signature s on H.

When selecting a set of generic elliptic curve domain parameters, the server SHOULD choose the smallest domain parameters it knows that are not smaller than the size the client requested. If the server does not know any domain parameters that are not smaller than the client request, then it MUST return the largest domain parameters it knows. In all cases, the size of the returned domain parameters SHOULD be at least 112 bits, and preferably at least 160 bits. Note that size of domain parameters is measured as the size in bits of the order of the generator.

2.2 Implementation

This key exchange algorithm is implemented with the following messages. The hash algorithm for computing the exchange hash is defined by the method name, and is called HASH. The key derivation function for computing the shared secret is defined by the method name, and is called KDF. The public key algorithm for signing is

negotiated with the KEXINIT messages.

1. The client sends the SSH_MSG_KEX_ECDH_REQUEST (Section 3.1) message. The value of curves is an ordered list of preferred elliptic curve domain parameters. In addition to the standardized named curves, there are two options for generic curves: generic-gfp and generic-gf2m. If curves contains either generic-gfp or generic-gf2m, the client MUST specify a range of acceptable curve sizes using the min, pref, and max values; otherwise, the values min, pref, and max MUST be 0. The following inequalities MUST be satisfied: min <= pref <= max.

2. The server responds with one of the following messages:

   * SSH_MSG_KEX_ECDH_CURVE_NAMED (Section 3.2)

   * SSH_MSG_KEX_ECDH_CURVE_GENERIC_GFP (Section 3.3)

   * SSH_MSG_KEX_ECDH_CURVE_GENERIC_GF2M (Section 3.4)

3. If the server responds with SSH_MSG_KEX_ECDH_CURVE_NAMED (Section 3.2), then the value of curve MUST be an element of the list curves, and MUST NOT be either generic-gfp or generic-gf2m.

4. If the server responds with either the SSH_MSG_KEX_ECDH_CURVE_GENERIC_GFP (Section 3.3) or SSH_MSG_KEX_ECDH_CURVE_GENERIC_GF2M (Section 3.4) message, then generic-gfp or generic-gf2m, respectively, MUST be in the list curves, and the size in bits of the order of the generator MUST be in the range min .. max inclusively, unless the server does not support any curves in that range, in which case the server MUST return the largest curve below that range that it supports.

5. The client sends the SSH_MSG_KEX_ECDH_INIT (Section 3.5) message.

6. The server responds with the SSH_MSG_KEX_ECDH_REPLY (Section 3.6) message.

3. Key Exchange Messages

The following message formats are defined in this document. The messages are used as described in Section 2.

3.1 SSH_MSG_KEX_ECDH_REQUEST

byte	SSH_MSG_KEX_ECDH_REQUEST
name-list	curves, a list of supported named curves and generic identifiers
uint32	min, minimal size in bits of acceptable generic elliptic curve domain parameters, or 0
uint32	pref, preferred size in bits of acceptable generic elliptic curve domain parameters, or 0
uint32	max, maximal size in bits of acceptable generic elliptic curve domain parameters, or 0

3.2 SSH_MSG_KEX_ECDH_CURVE_NAMED

byte	SSH_MSG_KEX_ECDH_CURVE_NAMED
string	curve, a named curve or generic identifier

The value of curve MUST NOT be either generic-gfp or generic-gf2m.

3.3 SSH_MSG_KEX_ECDH_CURVE_GENERIC_GFP

byte	SSH_MSG_KEX_ECDH_CURVE_GENERIC_GFP
mpint	prime, the prime defining the field
mpint	a, the curve parameter a
mpint	b, the curve parameter b
mpint	x, the x-coordinate of the generator
mpint	y, the y-coordinate of the generator
mpint	order, the order of the generator
uint32	cofactor, the cofactor of the generator
string	seed, the binary string used as a seed for random generation of the elliptic curve domain parameters

3.4 SSH_MSG_KEX_ECDH_CURVE_GENERIC_GF2M

```
byte       SSH_MSG_KEX_ECDH_CURVE_GENERIC_GF2M
mpint      irr, the irreducible polynomial defining the field
mpint      a, the curve parameter a
mpint      b, the curve parameter b
mpint      x, the x-coordinate of the generator
mpint      y, the y-coordinate of the generator
mpint      order, the order of the generator
uint32     cofactor, the cofactor of the generator
string     seed, the binary string used as a seed for random
           generation of the elliptic curve domain
           parameters
```

3.5 SSH_MSG_KEX_ECDH_INIT

```
byte       SSH_MSG_KEX_ECDH_INIT
mpint      c_x, the x-coordinate of the client's public key
mpint      c_y, the y-coordinate of the client's public key
```

3.6 SSH_MSG_KEX_ECDH_REPLY

```
byte       SSH_MSG_KEX_ECDH_REPLY
string     K_S, server public host key and certificates
mpint      s_x, the x-coordinate of the server's public key
mpint      s_y, the y-coordinate of the server's public key
string     s, the signature of H
```

The hash H is computed as the HASH hash of the concatenation of the following. Values marked with * are included in the concatenation only if the values are from messages that were actually transmitted.

For example, if the message SSH_MSG_KEX_ECDH_CURVE_NAMED (Section 3.2) is sent, then the only value marked with a * that will be included in the concatenation below is the value curve.

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
name-list	curves, a list of supported named curves and generic identifiers
uint32	min, minimal size in bits of acceptable generic elliptic curve domain parameters, or 0
uint32	pref, preferredsize in bits of acceptable generic elliptic curve domain parameters, or 0
uint32	max, maximal size in bits of acceptable generic elliptic curve domain parameters, or 0
string	curve*, a named curve or generic identifier
mpint	prime*, the prime defining the field
mpint	irr*, the irreducible polynomial defining the field
mpint	a*, the curve parameter a
mpint	b*, the curve parameter b
mpint	x*, the x-coordinate of the generator
mpint	y*, the y-coordinate of the generator
mpint	order*, the order of the generator
uint32	cofactor*, the cofactor of the generator
string	seed*, the binary string used as a seed for random generation of the elliptic curve domain parameters
mpint	c_x, the x-coordinate of the client's public key
mpint	c_y, the y-coordinate of the client's public key
mpint	s_x, the x-coordinate of the server's public key
mpint	s_y, the y-coordinate of the server's public key
mpint	K, the shared secret

3.7 Named Elliptic Curve Domain Parameters

The naming scheme for named elliptic curve domain parameters follows the naming conventions defined in Section 5 of [3].

Names that do not contain an at-sign (@) are reserved to be defined by IETF consensus (RFCs). Valid names of this form are listed in the following sections.

Anyone can define additional named elliptic curve domain parameters by using names in the format name@domainname, e.g. "our-curve@example.com". The format of the part preceding the at-sign is not specified. The part following the at-sign MUST be a valid fully qualified internet domain name controlled by the person or organization defining the curve name. It is up to each domain how it manages its local namespace.

3.7.1 Generic curve parameters

The following identifiers are used to indicate the use of an arbitrary generic curve over either GF(p) or GF(2^m).

```
generic-gfp   generic-gf2m
```

3.7.2 NIST-recommended curves

The following curves are defined in [8]. It is RECOMMENDED that an implementation support at least some of these curves.

```
nistp192   nistp224   nistp256   nistp384   nistp521
nistk163   nistb163   nistk233   nistb233   nistk283
nistb283   nistk409   nistb409   nistk571   nistb571
```

3.7.3 SEC-recommended curves

The following curves are defined in [9].

```
secp112r1   secp112r2   secp128r1   secp128r2   secp160k1
secp160r1   secp160r2   secp192k1   secp192r1   secp224k1
secp224r1   secp256k1   secp256r1   secp384r1   secp521r1

sect113r1   sect113r2   sect131r1   sect131r2   sect163k1
sect163r1   sect163r2   sect193r1   sect193r2   sect233k1
sect233r1   sect239k1   sect283k1   sect283r1   sect409k1
sect409r1   sect571k1   sect571r1
```

3.7.4 ANSI-recommended curves

The following curves are defined in [10].

```
prime192v1   prime192v2   prime192v3   prime239v1   prime239v2
prime239v3   prime256v1

c2pnb163v1   c2pnb163v2   c2pnb163v3   c2pnb176v1   c2tnb191v1
c2tnb191v2   c2tnb191v3   c2pnb208w1   c2tnb239v1   c2tnb239v2
c2tnb239v3   c2pnb272w1   c2pnb304w1   c2tnb359v1   c2pnb368w1
c2tnb431r1
```

3.7.5 WTLS-recommended curves

The following curves are defined in [11].

```
wtls1    wtls3    wtls4    wtls5    wtls6    wtls7    wtls8
wtls9    wtls10   wtls11   wtls12
```

3.8 Summary of Message Numbers

The following message numbers have been defined in this document:

```
#define SSH_MSG_KEX_ECDH_REQUEST             30
#define SSH_MSG_KEX_ECDH_CURVE_NAMED         31
#define SSH_MSG_KEX_ECDH_CURVE_GENERIC_GFP   32
#define SSH_MSG_KEX_ECDH_CURVE_GENERIC_GF2M  33
#define SSH_MSG_KEX_ECDH_INIT                34
#define SSH_MSG_KEX_ECDH_REPLY               35
```

Figure 14: Summary of Message Numbers

The numbers 30-49 are key exchange-specific and may be redefined by other key exchange methods.

4. ECDH Key Exchange Methods

4.1 ecdh-exchange-sha1

The "ecdh-exchange-sha1" method specifies Elliptic Curve Diffie-Hellman Parameter and Key Exchange with SHA-1 as HASH and ECDH without cofactor multiplication as KDF. The SHA-1 hash algorithm is defined in [12].

In ECDH without cofactor multiplication, the shared secret is generated as follows. Let k be the private key, and P be the generator. Then the shared secret K is the x-coordinate of the affine representation of the point Q = k * P.

4.2 ecdhc-exchange-sha1

The "ecdhc-exchange-sha1" method specifies Elliptic Curve Diffie-Hellman Parameter and Key Exchange with SHA-1 as HASH and ECDH with cofactor multiplication as KDF. The SHA-1 hash algorithm is defined in [12].

In ECDH with cofactor multiplication, the shared secret is generated as follows. Let k be the private key, P be the generator, and h be the cofactor. Then the shared secret K is the x-coordinate of the affine representation of the point Q = (k * h) * P.

5. Security Considerations

The Elliptic Curve Diffie-Hellman key agreement algorithm is defined in [6] and [7]. The appropriate security considerations of those documents apply.

The key exchange methods defined in Section 4 rely on the SHA-1 hash algorithm as defined in [12]. The appropriate security considerations of that document apply.

Server implementations that generate random elliptic curve domain parameters SHOULD generate those parameters verifiably at random, using for example the algorithms defined in section A.16.7 of [6]. Client implementations SHOULD verify that generic parameters have been generated randomly, using for example the verification algorithms defined in section A.16.8 of [6].

Since ECDH allows 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 and bulk encryption algorithms must be chosen appropriately. Information regarding estimated equivalence of key sizes is available in [2].

For most applications, it should be sufficient to use an existing named curve as recommended in Section 3.7.

6. Intellectual Property Rights

The IETF has been notified of intellectual property rights claimed in regard to the specification contained in this document. For more information, consult the online list of claimed rights (http://www.ietf.org/ipr.html).

The IETF takes no position regarding the validity or scope of any intellectual property 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; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in [13]. Copies of claims of rights made available for publication 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 Secretariat.

7. Acknowledgments

The author wishes to thank Sheueling Chang and Vipul Gupta of Sun Microsystems. This work was largely performed during a student internship at Sun Microsystems Laboratories from the University of Waterloo.

References

[1] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[2] Lenstra, A. and E. Verheul, "Selecting Cryptographic Key Sizes", Journal of Cryptology 14 (2001) 255-293, <http://www.cryptosavvy.com>.

[3] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Protocol Architecture", draft-ietf-secsh-architecture-15.txt (work in progress), October 2003.

[4] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Transport Layer Protocol", draft-ietf-secsh-transport-17.txt (work in progress), October 2003.

[5] SECG, "Elliptic Curve Cryptography", SEC 1, 2000, <http://www.secg.org/>.

[6] IEEE, "Standard Specifications for Public Key Cryptography", IEEE 1363, 2000.

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

[8] NIST, "Recommended Elliptic Curves for Federal Government Use", July 1999, <http://csrc.nist.gov/csrc/fedstandards.html>.

[9] SECG, "Recommended Elliptic Curve Domain Parameters", SEC 2, 2000, <http://www.secg.org/>.

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

[11] WAP, "Wireless Transport Layer Security", WAP 261-WTLS-20010406-a, April 2001.

[12] NIST, "Secure Hash Standard", FIPS 180-2, 2002.

[13] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", RFC 2028, BCP 11, October 1996.

Author's Address

Douglas Stebila
Sun Microsystems Laboratories
2600 Casey Avenue
MS UMTV29-112
Mountain View, CA 94043
USA

EMail: douglas.stebila@sun.com

Full Copyright Statement

Acknowledgement

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