Transport Layer Security Working Group
INTERNET-DRAFT
Expires September, 1998

T. Dierks
Consensus Development Corp.
B. Anderson
Certicom Corp.
March 13, 1998

ECC Cipher Suites For TLS
draft-ietf-tls-ecc-00.txt

1. Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as work in progress.

To learn the current status of any Internet-Draft, please check the 1id-abstracts.txt listing contained in the Internet Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

2. Introduction

This document describes additions to TLS to support the Elliptic Curve Cryptosystem (ECC). The document assumes that the reader is familiar with the TLS protocol.

The document defines cipher suites which use the Elliptic Curve Encryption Scheme (ECES), the Elliptic Curve Digital Signature Algorithm (ECDSA), the Elliptic Curve Nyberg-Rueppel Signature Scheme with Appendix (ECNRA), the Elliptic Curve Diffie-Hellman Key Agreement (ECDH), and the Elliptic Curve Menezes-Qu-Vanstone Key Agreement (ECMQV) key establishment algorithms. References to these algorithms can be found in section 13.

3. Table of Contents

1. Status of this Memo 1
2. Introduction 1
3. Table of Contents 2
4. Rationale 2
5. Elliptic Curve Key Establishment Methods 3
6. Key Establishment Operation 5
6.1. ECES_ECDSA 6
6.2. ECES_ECNRA 6
6.3. ECDHE_ECDSA 6
6.4. ECDHE_ECDSA_EXPORT 7
6.5. ECDHE_ECNRA 7
6.6. ECDHE_ECNRA_EXPORT 7
6.7. ECDH_ECDSA 7
6.8. ECDH_ECNRA 8
6.9. ECMQV_ECDSA 8
6.10. ECMQV_ECNRA 9
6.11. ECDH_anon 9
6.12. ECDH_anon_EXPORT 9
7. Client Certification 9
8. Data Structures 10
8.1. Server Key Exchange 10
8.2. Certificate Request 13
8.3. Client Key Exchange 14
8.4. Certificate Verify 15
9. Elliptic Curve Certificates 15
10. Cipher Suites 16
11. Elliptic Curve Cryptography Definitions 17
12. Recommended Cipher Suites 17
13. References 17
14. Security Considerations 18
15. Authors' Addresses 18

4. Rationale

Several design goals drove our choice of key establishment algorithms:

1. A desire to replicate all of the functionality and operating modes found in the current TLS cipher suites based on integer factorization and discret log cryptographic algorithms.

2. While we wished to define cipher suites which use export-strength

cryptography, we did not want to define any cipher suites which would require certificates with export-strength keys; thus, exportable cipher suites are only defined for those key establishment mechanisms which use the certificate key for authentication rather than for key establishment.

These criteria for key establishment algorithms, when combined with a number of symmetric algorithms, led to a large number of possible cipher suites. This is problematic in that it could lead to a lack of interoperability due to implementors supporting different subsets of the available cipher suites. In order to alleviate this, we have indicated two of the total cipher suites as recommended (see section 12). Unless there are specific reasons to choose other cipher suites, implementors should implement the recommended suites first.

5. Elliptic Curve Key Establishment Methods

Key establishment is the terminology used in ISO standards to refer to the methods of establishing a shared key between two or more parties. Within key establishment there are two classifications: The operation is called key transport when only one party contributes to the generation of the shared key. The operation is called key agreement when 2 or more parties contribute to the generation of the shared key. For the purposes of this definition, the key in question is the premaster secret: TLS uses the master secret generation process to ensure that both parties contribute to the eventual master secret.

The cipher suites defined here use the following key establishment methods:

ECES_ECDSA Elliptic-curve encryption is used for the key transport; the server's certificate is signed using ECDSA.

ECES_ECNRA Elliptic-curve encryption is used for the key

transport; the server's certificate is signed using ECNRA.

ECDHE_ECDSA Ephemeral elliptic-curve Diffie-Hellman is used for the key agreement; the server signs the parameters with an ECDSA key and is authenticated with a certificate signed with ECDSA.

ECDHE_ECDSA_EXPORT Ephemeral elliptic-curve Diffie-Hellman in export strength is used for the key agreement; the server signs the parameters with an ECDSA key and is authenticated with a certificate signed with ECDSA.

ECDHE_ECNRA Ephemeral elliptic-curve Diffie-Hellman is used for the key agreement; the server signs the parameters with an ECNRA key and is authenticated with a certificate signed with ECNRA.

ECDHE_ECNRA_EXPORT Ephemeral elliptic-curve Diffie-Hellman in export strength is used for the key agreement; the server signs the parameters with an ECNRA key and is authenticated with a certificate signed with ECNRA.

ECDH_ECDSA Fixed elliptic-curve Diffie-Hellman is used for the key agreement; the server's certificate is signed with ECDSA.

ECDH_ECNRA Fixed elliptic-curve Diffie-Hellman is used for the key agreement; the server's certificate is signed with ECNRA.

ECMQV_ECDSA Ephemeral elliptic-curve MQV is used for key agreement and authentication; the server is authenticated with a certificate signed with

ECDSA.

ECMQV_ECNRA — Ephemeral elliptic-curve MQV is used for key agreement and authentication; the server is authenticated with a certificate signed with ECNRA.

ECDH_anon — Anonymous elliptic-curve Diffie-Hellman is used for the key agreement.

ECDH_anon_EXPORT — Anonymous elliptic-curve Diffie-Hellman in export strength is used for the key agreement.

Key establishment mechanisms which indicate that they are for export strength should use an ECC key for the key agreement of no more than 113 bits. A 113-bit ECC key provides security that is roughly equivalent to a 512-bit RSA key and is expected to be eligible for export. The following table relates ECC key sizes to RSA key sizes

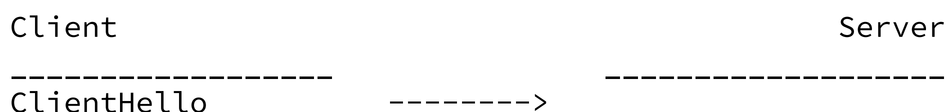
of equivalent security. These key sizes are considered equivalent in terms of work factor required to recover the private key using currently known fastest methods for solving the underlying mathematical problems of ECC and RSA.

ECC	RSA	Time to break (MIPS-years)
106 bits	512 bits	1E4 MY
132 bits	768 bits	1E8 MY
160 bits	1024 bits	1E11 MY
191 bits	1536 bits	1E14 MY
211 bits	2048 bits	1E20 MY

Table 1: ECC and RSA key sizes for equivalent security

6. Key Establishment Operation

The TLS key establishment protocol involves this message exchange:



```
Client                                              Server
------------------                          -------------------
ClientHello          -------->
```

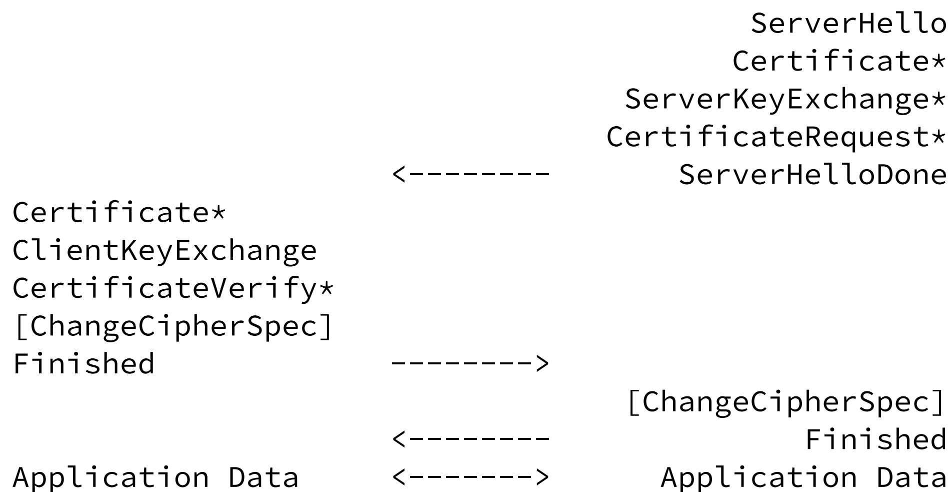


Figure 1: Message flow in a full TLS handshake
* - Message is not sent under some conditions

Of these messages, the ones involved in the key establishment itself are the server's Certificate message, the ServerKeyExchange, the client's Certificate message, and the ClientKeyExchange.

In order to specify the ECC cipher suites, we must specify the following elements for each key establishment algorithm:

- The format of the server's certificate.

- The format of the server key exchange message

- For methods in which the client's certificate can participate in the key agreement, the format of the client's certificate and the criteria for deciding if this certificate is eligible to participate in the key agreement.

- The format of the client key exchange message

- How to arrive at the premaster secret given all the preceding information.

Several different key establishment modes are available. In order to allow full negotiation of supported algorithms, the signature algorithm used for the server's X.509 certificate is encoded into the

cipher suite for those key establishment mechanisms where no signature algorithm is used; for those key establishments which utilize signature algorithms, the certificate signature algorithm is expected to be the same as the algorithm used in the key establishment.

6.1. ECES_ECDSA

In ECES_ECDSA, the server sends a certificate with an ECES-capable public key in it. The server's certificate should be signed with ECDSA. A ServerKeyExchange message is not sent because the server's certificate contains all the necessary keying information for the client to complete the key establishment.

The client's certificate is not involved in the key establishment for this method, although the client can still be authenticated via the normal mechanism.

The client generates a 48 byte premaster secret, encrypts it using ECES using the public key from the server's certificate, and sends it to the server in the ClientKeyExchange message (see section 8.3).

This premaster secret is decrypted by the server and both sides use it to generate the master secret for this TLS session.

6.2. ECES_ECNRA

ECES_ECNRA is the same as ECES_ECDSA except for the fact that the server's certificate is signed by its CA with ECNRA.

6.3. ECDHE_ECDSA

In ECDHE_ECDSA, the server's certificate has an ECDSA key and is, in

turn, signed by its CA with ECDSA. The ServerKeyExchange message contains an ephemeral ECDH key and a specification of the curve for this key. (see section 8.1). These parameters are signed with the server's authenticated ECDSA key.

The client's certificate is not involved in the key establishment for this method, although the client can still be authenticated via the normal mechanism.

The client should verify the signature on the ServerKeyExchange message and generate an ECDH key on the same curve as the server's ephemeral key. The client encodes the public half of that key into the ClientKeyExchange message and sends it to the server.

The client and server perform an ECDH key agreement using their private keys and the public keys they have sent to each other. The resultant shared secret is the premaster secret.

6.4. ECDHE_ECDSA_EXPORT

ECDHE_ECDSA_EXPORT is the same as ECDHE_ECDSA except for the fact that the curve used for the server's ephemeral ECDH key should be no longer than 113 bits. Because the server's certified key is only used for authentication, its length is unrestricted.

6.5. ECDHE_ECNRA

ECDHE_ECNRA is the same as ECDHE_ECDSA except for the fact that the server's public key is an ECNRA key and the server's certificate is signed by its CA with ECNRA.

6.6. ECDHE_ECNRA_EXPORT

ECDHE_ECNRA_EXPORT is the same as ECDHE_ECNRA except for the fact that the curve used for the server's ephemeral ECDH key should be no longer than 113 bits. Because the server's certified key is only used for authentication, its length is unrestricted.

6.7. ECDH_ECDSA

In ECDH_ECDSA, the server's certificate contains an ECDH public key. This certificate is signed by the server's CA using ECDSA. The ServerKeyExchange message is not sent because the server's certificate contains all the necessary keying information for the client to complete the key establishment.

If the server requests client authentication and includes the ecdsa_fixed_dh or ecnra_fixed_dh client certificate types (see

section 8.2) and the client has a certificate which contains an ECDH

key on the same curve as the server's public key, and this certificate is otherwise eligible to be used for client authentication, then the client's certified public key is used in conjunction with the server's public key to do an ECDH key agreement; the resultant shared secret is the premaster key. In this situation, the client key exchange message is empty when sent and the client CertificateVerify message is not sent, as both the client and the server are authenticated by their ability to arrive at the same premaster secret.

If client certification is not requested or if the client does not have a certificate with a suitable ECDH public key, the client can generate an ephemeral key on the same curve as the server's public key. This key is encoded into the ClientKeyExchange message (see section 8.3) and used in conjunction with the server's key to complete the ECDH key agreement, yielding the premaster secret.

6.8. ECDH_ECNRA

ECDH_ECNRA is the same as ECDH_ECDSA except for the fact that the server's certificate is signed by its CA with ECNRA.

6.9. ECMQV_ECDSA

In ECMQV_ECDSA, the server's certificate contains an ECMQV key and is signed by the server's CA with ECDSA. The server then generates an temporary key pair and sends the public half of the temporary key in the ServerKeyExchange message (see section 8.1).

If the server requests client authentication and includes the ecdsa_mqv or ecnra_mqv client certificate types (see section 8.2) and the client has a certificate which contains an ECMQV key on the same curve as the server's public key, and this certificate is otherwise eligible to be used for client authentication, then the client should send that certificate, then generate a temporary key and send the public half of that key in the ClientKeyExchange message (see section 8.3). The client and server then perform an MQV key agreement using their private keys and their peer's public keys (for each party, both the certified and temporary key pairs are used). The resultant shared secret is the premaster secret. The client CertificateVerify message is not sent, as both the client and the server are authenticated by their ability to arrive at the same premaster secret.

If client certification is not requested or if the client does not have a certificate with a suitable ECMQV public key, the client should generate two temporary key pairs on the same curve as the

server's public key. The public halves of these temporary key pairs are encoded into the ClientKeyExchange message. One key pair is the usual temporary key used for MQV and the other takes the place of the certified key. Each side performs an MQV key agreement using the peer's public keys and its own private keys, yielding the premaster secret.

6.10. ECMQV_ECNRA

ECMQV_ECNRA is the same as ECMQV_ECDSA except for the fact that the server's certificate is signed by its CA with ECNRA.

6.11. ECDH_anon

In ECDH_anon, an anonymous Elliptic-Curve Diffie-Hellman operation is used to arrive at the premaster secret. In this case, the server is not authenticated and may not request that the client authenticate itself. The server's Certificate message is not sent. The ServerKeyExchange message contains the specification of a curve and a Diffie-Hellman public key (see section 8.1). The client responds with a ClientKeyExchange message containing a Diffie-Hellman public key on the same curve; the premaster secret is the shared secret resulting from an Elliptic Curve Diffie-Hellman key agreement with these keys.

6.12. ECDH_anon_EXPORT

ECDH_anon_EXPORT is the same as ECDH_anon except for the fact that the curve used for the server's ephemeral ECDH key should be no longer than 113 bits.

7. Client Certification

Six new client certificate types have been added: ecdsa_sign, ecnra_sign, ecdsa_fixed_dh, ecnra_fixed_dh, ecdsa_mqv, and ecnra_mqv. As noted above, the fixed_dh and mqv types are used in key establishment methods which allow the client's certified key to participate in key agreement. In these cases, the CertificateVerify message is not sent; the client's ability to arrive at the same premaster secret as the server demonstrates its control over the private half of the certified public key.

One of these certificates is eligible for use in the key agreement operation if it has a key which can be used with that algorithm. Because elliptic curve keys have the same mathematical properties for all the algorithms discussed in this specification, a certificate

could have a key which was authorized for use in any of several algorithms or for only a particular algorithm. In addition to the

key's eligibility, it must be defined using the same curve parameters as the server's key to be used in a operation with it. Of course, the use of a certificate is always subject to any and all policy constraints placed on it.

In these certificates, the ecdsa or ecnra refers to the algorithm which the CA uses to sign the client's certificate.

The ecdsa_sign and ecnra_sign certificate types are used in other key establishment methods and in cases where the client can not or chooses not to supply a suitable certificate to participate in one of the above methods. In these cases, the client must send a CertificateVerify message to demonstrate its control of the private half key of the certified key pair. (See section 8.4).

Certificates requested with the ecdsa_sign ClientCertificateType must include an ECDSA public key and be signed by the CA with ECDSA; ecnra_sign certificates must include an ECNRA key and be signed with ECNRA.

With all key establishment methods, it is permissible to request a client certificate using a different algorithm than the one used for the server's certificate; for example, a server doing a ECDHE_ECDSA or ECMQV_ECDSA key establishment could still request an ECNRA client certificate.

8. Data Structures

Here the descriptions of the data structures exchanged are given. The presentation language is the same as that used in the TLS specification. Because these specifications extend the TLS protocol specification, these descriptions should be merged with those in TLS and in any other specifications which extend TLS. This means that enum types may not specify all the possible values and structures with multiple formats chosen with a select() clause may not indicate all the possible cases.

8.1. Server Key Exchange

This messages is sent in the following key establishment methods:

```
ECDHE_ECDSA
ECDHE_ECDSA_EXPORT
ECDHE_ECNRA
ECDHE_ECNRA_EXPORT
ECMQV_ECDSA
ECMQV_ECNRA
ECDH_anon
```



```
ECDH_anon_EXPORT
```

It can contain elliptic curve Diffie-Hellman keys, either signed or unsigned, or MQV parameters.

Structure of this message:

```
enum { ec_eces,
       ec_diffie_hellman,
       ec_menezes_qu_vanstone } KeyExchangeAlgorithm;

enum { ec_prime_p (1),
       ec_characteristic_two (2), (255) } ECFieldID;

enum { ec_basis_onb, ec_basis_trinomial,
       ec_basis_pentanomial } ECBasisType;

struct {
    opaque a <1..2^8-1>;
    opaque b <1..2^8-1>;
    opaque seed <0..2^8-1>;
} ECCurve;
```

a, b: These parameters specify the coefficients of the elliptic curve. Each value shall be the octet string representation of a field element following the conversion routine in [X9.62], section 4.3.1.

seed: This is an optional parameter used to derive the coefficients of a randomly generated elliptic curve.

```
    struct {
        opaque point <1..2^8-1>;
    } ECPoint;
```

point: This is the octet string representation of an elliptic curve point following the conversion routine in [X9.62], section 4.4.2.a. The representation format is defined following the definition in [X9.62], section 4.4.

```
    struct {
        ECFieldID   field;
        select (field) {
            case ec_prime_p:
                opaque      prime_p <1..2^8-1>;
            case ec_characteristic_two:
```



```
                uint16      m;
                ECBasisType basis;
                select (basis) {
                    case ec_basis_onb:
                        struct { };
                    case ec_trinomial:
                        opaque  k <1..2^8-1>;
                    case ec_pentanomial:
                        opaque  k1 <1..2^8-1>;
                        opaque  k2 <1..2^8-1>;
                        opaque  k3 <1..2^8-1>;
                };
        };
        ECCurve     curve;
        ECPoint     base;
        opaque      order <1..2^8-1>;
        opaque      cofactor <1..2^8-1>;
    } ECParameters;
```

field: This identifies the finite field over which the elliptic curve is defined.

prime_p: This is the odd prime defining the field Fp.

m: This is the degree of the characteristic-two field F2^m.

k: The exponent k for the trinomical basis representation x^m + x^k + 1.

k1, k2, k3: The exponents for the pentanomial representation x^m + x^k3 + x^k2 + x^k1 + 1.

curve: Specifies the coefficients a and b of the elliptic curve E.

base: The base point P on the elliptic curve.

order: The order n of the base point. The order of a point P is the smallest possible integer n such that nP = 0 (the point at infinity).

cofactor: The integer h = #E(Fq)/n, where #E(Fq) represents the number of points on the elliptic curve E defined over the field Fq.

```
    struct {
        ECParameters    curve_params;
        ECPoint         public;
    } ServerECDHParams;
```


curve_params: This specifies the curve on which the elliptic-curve Diffie-Hellman key agreement is to occur.

public: The ephemeral public key for the elliptic-curve Diffie-Hellman key agreement.

```
    struct {
        ECPoint         temp_public;
    } ServerMQVParams;
```

temp_public: The temporary MQV public key; the curve on which the MQV operation will take place is specified by the server's certificate.

```
    enum { ec_dsa, ec_nra } SignatureAlgorithm;

    select (SignatureAlgorithm) {
        case ec_dsa:
```

```
                digitally-signed struct {
                    opaque sha_hash[20];
                };
            case ec_nra:
                digitally-signed struct {
                    opaque sha_hash[20];
                };
        } Signature;

        select (KeyExchangeAlgorithm) {
            case ec_diffie_hellman:
                ServerECDHParams    params;
                Signature           signed_params;
            case ec_menezes_qu_vanstone:
                ServerMQVParams     params;
        } ServerKeyExchange;
```

Note: The anonymous case for Signature is used for ECDH_anon and ECDH_anon_EXPORT key establishment methods: in this case, the Signature element is empty.

8.2. Certificate Request

The only addition to this message is six new types for the client certificate.

Structure of this message:

```
        enum {
```



```
            ecdsa_sign(5), ecnra_sign(6),
            ecdsa_fixed_dh(7), ecnra_fixed_dh(8),
            ecdsa_mqv (9), ecnra_mqv (10), (255)
        } ClientCertificateType;
```

8.3. Client Key Exchange

This message is sent in all key exchanges. It can contain either an ECES encrypted secret, an ECDH public key (for use in ECDHE or ECDH_anon key establishment methods), an ECMQV temporary public key,

or two temporary keys for use with MQV when the client does not have a suitable certificate.

Structure of this message:

```
struct {
    select (PublicValueEncoding) {
        case implicit: struct { };
        case explicit: ECPoint  ecdh_Yc;
    } ecdh_public;
} ClientECDiffieHellmanPublic;
```

If the client has sent a certificate with an ECDH key, the PublicValueEncoding will be implicit and this message will be empty. Otherwise, ecdh_Yc will be the client's public value for the Diffie-Hellman key agreement.

```
struct {
    select (PublicValueEncoding) {
        case implicit: struct { };
        case explicit: ECPoint  ecmqv;
    } ecmqv_public;
    ECPoint     ecmqv_temp;
} ClientECMQVPublic;
```

If the client has sent a certificate with an MQV key, the PublicValueEncoding will be implicit and the ecmqv_public field will be empty; otherwise, ecmqv will contain the client's MQV public value. In either case, ecmqv_temp will contain the temporary public key for the MQV operation.

In the explicit case, the cost of an additional key generation can be saved by generating only one ephemeral key and sending two copies: one in ecmqv and one in ecmqv_temp.


```
struct {
    select (KeyExchangeAlgorithm) {
        case ec_eces: EncryptedPreMasterSecret;
        case ec_diffie_hellman: ClientECDiffieHellmanPublic;
```

```
            case ec_menezes_qu_vanstone: ClientECMQVPublic;
        } exchange_keys;
    } ClientKeyExchange;
```

In the ECES case, the premaster secret will be sent encrypted with the server's public key. The standard TLS definition of EncryptedPreMasterSecret is suitable for this transmission.

8.4. Certificate Verify

This message is sent when the client has sent a certificate which did not participate in a Diffie-Hellman or Menezes-Qu-Vanstone key agreement.

This type needs no new definition: the CertificateVerify message in TLS uses the Signature type, which we have extended for ECDSA and ECNRA (see section 8.1).

9. Elliptic Curve Certificates

All X.509 certificates must be in compliance with the PKIX profile of the X.509 standard [PKIX]. Elliptic curve keys should be encoded into X.509 certificates as specified in [PKIX-ECDSA]. However, this document currently only specifies formats for ECDSA keys and signatures.

When this document refers to a certificate with an ECDSA, ECNRA, ECES, ECDH, or ECMQV key, it means a public key which is capable of performing a particular algorithm and which is permitted by the policy encoded in the certificate to participate in this algorithm. This may be a key which is specifically indicated as being useful for a particular algorithm or a general-purpose elliptic curve key which is allowed to perform a particular operation.

The X.509 key usage extension encodes functions a key is allowed to perform. The relevant key usage bits for algorithms are:

Algorithm	Key Usage Bit
ECDSA	digitalSignature
ECNRA	digitalSignature
ECES	keyEncipherment
ECDH	keyAgreement
ECMQV	keyAgreement

Table 2: Pertinent X.509 key usage bits

A TLS entity shall not present a certificate which is not eligible to participate in the negotiated cipher suite and shall refuse to communicate with a TLS peer which presents such a certificate.

10. Cipher Suites

The following cipher suites are defined:

```
CipherSuite TLS_ECES_ECDSA_NULL_SHA                      = { 0x00, 0x2C }
CipherSuite TLS_ECES_ECDSA_WITH_RC4_128_SHA              = { 0x00, 0x2D }
CipherSuite TLS_ECES_ECDSA_WITH_DES_CBC_SHA              = { 0x00, 0x2E }
CipherSuite TLS_ECES_ECDSA_WITH_3DES_EDE_CBC_SHA         = { 0x00, 0x2F }
CipherSuite TLS_ECES_ECNRA_NULL_SHA                      = { 0x00, 0x30 }
CipherSuite TLS_ECES_ECNRA_WITH_RC4_128_SHA              = { 0x00, 0x31 }
CipherSuite TLS_ECES_ECNRA_WITH_DES_CBC_SHA              = { 0x00, 0x32 }
CipherSuite TLS_ECES_ECNRA_WITH_3DES_EDE_CBC_SHA         = { 0x00, 0x33 }
CipherSuite TLS_ECDHE_ECDSA_NULL_SHA                     = { 0x00, 0x34 }
CipherSuite TLS_ECDHE_ECDSA_WITH_RC4_128_SHA             = { 0x00, 0x36 }
CipherSuite TLS_ECDHE_ECDSA_WITH_DES_CBC_SHA             = { 0x00, 0x37 }
CipherSuite TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA        = { 0x00, 0x38 }
CipherSuite TLS_ECDHE_ECDSA_EXPORT_WITH_DES40_CBC_SHA = { 0x00, 0x39 }
CipherSuite TLS_ECDHE_ECDSA_EXPORT_WITH_RC4_40_SHA       = { 0x00, 0x40 }
CipherSuite TLS_ECDHE_ECNRA_NULL_SHA                     = { 0x00, 0x41 }
CipherSuite TLS_ECDHE_ECNRA_WITH_RC4_128_SHA             = { 0x00, 0x42 }
CipherSuite TLS_ECDHE_ECNRA_WITH_DES_CBC_SHA             = { 0x00, 0x43 }
CipherSuite TLS_ECDHE_ECNRA_WITH_3DES_EDE_CBC_SHA        = { 0x00, 0x44 }
CipherSuite TLS_ECDHE_ECNRA_EXPORT_WITH_DES40_CBC_SHA = { 0x00, 0x45 }
CipherSuite TLS_ECDHE_ECNRA_EXPORT_WITH_RC4_40_SHA       = { 0x00, 0x46 }
CipherSuite TLS_ECDH_ECDSA_NULL_SHA                      = { 0x00, 0x47 }
CipherSuite TLS_ECDH_ECDSA_WITH_RC4_128_SHA              = { 0x00, 0x48 }
CipherSuite TLS_ECDH_ECDSA_WITH_DES_CBC_SHA              = { 0x00, 0x49 }
CipherSuite TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA         = { 0x00, 0x4A }
CipherSuite TLS_ECDH_ECNRA_NULL_SHA                      = { 0x00, 0x4B }
CipherSuite TLS_ECDH_ECNRA_WITH_RC4_128_SHA              = { 0x00, 0x4C }
CipherSuite TLS_ECDH_ECNRA_WITH_DES_CBC_SHA              = { 0x00, 0x4D }
CipherSuite TLS_ECDH_ECNRA_WITH_3DES_EDE_CBC_SHA         = { 0x00, 0x4E }
CipherSuite TLS_ECMQV_ECDSA_NULL_SHA                     = { 0x00, 0x4F }
CipherSuite TLS_ECMQV_ECDSA_WITH_RC4_128_SHA             = { 0x00, 0x50 }
CipherSuite TLS_ECMQV_ECDSA_WITH_DES_CBC_SHA             = { 0x00, 0x51 }
CipherSuite TLS_ECMQV_ECDSA_WITH_3DES_EDE_CBC_SHA        = { 0x00, 0x52 }
CipherSuite TLS_ECMQV_ECNRA_NULL_SHA                     = { 0x00, 0x53 }
CipherSuite TLS_ECMQV_ECNRA_WITH_RC4_128_SHA             = { 0x00, 0x54 }
CipherSuite TLS_ECMQV_ECNRA_WITH_DES_CBC_SHA             = { 0x00, 0x55 }
CipherSuite TLS_ECMQV_ECNRA_WITH_3DES_EDE_CBC_SHA        = { 0x00, 0x56 }
CipherSuite TLS_ECDH_anon_NULL_WITH_SHA                  = { 0x00, 0x57 }
```

```
CipherSuite TLS_ECDH_anon_WITH_RC4_128_SHA           = { 0x00, 0x58 }
```



```
CipherSuite TLS_ECDH_anon_WITH_DES_CBC_SHA           = { 0x00, 0x59 }
CipherSuite TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA      = { 0x00, 0x5A }
CipherSuite TLS_ECDH_anon_EXPORT_WITH_DES40_CBC_SHA  = { 0x00, 0x5B }
CipherSuite TLS_ECDH_anon_EXPORT_WITH_RC4_40_SHA     = { 0x00, 0x5C }
```

Table 3: TLS ECC cipher suites

The key establishment method, cipher, and hash algorithm for each cipher suite are easily determined by examining the name. Those cipher suites which use the "NULL" cipher or one of the "EXPORT" key establishment mechanisms are considered to be "exportable" cipher suites for the purposes of the TLS protocol.

11. Elliptic Curve Cryptography Definitions

These definitions provide a quick reference for the elliptic curve terms.

Elliptic curve	Definition to come.
Elliptic curve point	Definition to come.
EC parameters	Definition to come.
EC private key	Definition to come.
EC public key	Definition to come.
EC key pair	Definition to come.

12. Recommended Cipher Suites

In order to promote common interoperability, two cipher suites are recommended for initial implementation: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA and TLS_ECDHE_ECDSA_EXPORT_WITH_RC4_40_SHA. Implementing these two gives a basis of cryptographic strength, perfect forward secrecy, and well-accepted algorithms.

13. References

[ECDH] IEEE P1363 Working Draft, February, 1997.

[ECDSA] IEEE P1363 Working Draft, February, 1997.

[ECDSA] ANSI X9.62 Working Draft, November 17, 1997.

[ECES] ANSI X9.63 Working Draft.

[ECMQV] IEEE P1363 Working Draft, February, 1997.

[ECNRA] IEEE P1363 Working Draft, February, 1997.

[PKIX] R. Housley, W. Ford, W. Polk, D. Solo, Internet Public Key Infrastructure: Part I: X.509 Certificate and CRL Profile, <draft-ietf-pkix-ipki-part1-06.txt>, October 1997.

[PKIX-ECDSA] L. Bassham, D. Johnson, W. Polk, Representation of Elliptic Curve Digital Signature Algorithm (ECDSA) Keys and Signatures in Internet X.509 Public Key Infrastructure Certificates <draft-ietf-pkix-ipki-ecdsa-01.txt>, November 1997.

[X9.62] ANSI X9.62 Working Draft, November 17, 1997.

14. Security Considerations

This document is entirely concerned with security mechanisms. Implementors should take care to ensure that code which controls security mechanisms is free of errors which might be exploited by attackers.

15. Authors' Addresses

Authors:

Tim Dierks
Consensus Development
timd@consensus.com

Bill Anderson

Certicom
banderson@certicom.com

Contributors:

Gilles Garon
ggaron@aol.com