TLS Working Group
INTERNET-DRAFT
Expires: September 14, 2001

Simon Blake-Wilson
Tim Dierks
Chris Hawk
Certicom Corp.
15 March 2001

ECC Cipher Suites for TLS

<draft-ietf-tls-ecc-01.txt>

Status of this Memo

Abstract

This document describes additions to TLS to support Elliptic Curve Cryptography (ECC). In particular it defines new key exchange algorithms which use the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Diffie-Hellman Key Agreement Scheme (ECDH), and it defines how to perform client authentication with ECDSA and ECDH.

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 [MUST].

Please send comments on this document to the TLS mailing list.

Table of Contents

1. Introduction 2
2. Elliptic Curve Key Exchange Algorithms 4
2.1. ECDH_ECDSA 5
2.2. ECDH_ECDSA_EXPORT 5
2.3. ECDH_RSA 6
2.4. ECDH_RSA_EXPORT 6
2.5. ECDH_anon 6
2.6. ECDH_anon_EXPORT 6
3. ECC Client Authentication 7
3.1. ECDSA_sign 7
3.2. ECDSA_fixed_ECDH 8
3.3. RSA_fixed_ECDH 9
4. Data Structures and Computations 9
4.1. Server Certificate 10
4.2. Server Key Exchange 11
4.3. Certificate Request 15
4.4. Client Certificate 15
4.5. Client Key Exchange 16
4.6. Certificate Verify 18
4.7. Computing the Master Secret 19
5. Cipher Suites 19
6. Security Considerations 20
7. Intellectual Property Rights 20
8. Acknowledgments 21
9. References 21
10. Authors' Addresses 22

1. Introduction

This document describes additions to TLS to support Elliptic Curve Cryptography (ECC). In particular, it defines:

- new key exchange algorithms which use the Elliptic Curve Digital Signature Algorithm (ECDSA), and the Elliptic Curve Diffie-Hellman Key Agreement Scheme (ECDH); and

- new client authentication methods which use ECDSA and ECDH.

In order to enable the use of these features within TLS, the document defines enhanced data structures to convey the information that the mechanisms need to exchange, the computational procedures involved in the operation of the mechanisms, and new cipher suites based on the mechanisms.

Use of ECC within TLS may provide both bandwidth and computational savings compared to other public-key cryptographic techniques. Furthermore, the efficiencies provided by ECC may increase as security requirements increase based on Moore's law - this is illustrated by the following table, based on [LEN], which gives approximate comparable key sizes for symmetric systems, ECC systems, and DH/DSA/RSA systems based on the running times of the best algorithms known today.

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

Table 1: Comparable key sizes (in bits)

The savings that ECC may offer are likely to become increasingly desirable with the widespread use of TLS by wireless devices - discussed, for example, in [TLS-EXT].

This document assumes the reader is familiar with both ECC and TLS. ECC is described in ANSI X9.62 [ANSIX962], FIPS 186-2 [FIPS186-2], IEEE 1363 [IEEE1363], and SEC 1 [SEC1]. TLS is described in RFC 2246 [TLS].

The choice of mechanisms included in this document was motivated by a desire to provide mechanisms which are secure, which are as efficient as possible, and which are capable of replicating all of the functionality and operating modes found in the existing TLS mechanisms based on integer factorization and discrete logarithm cryptographic systems. TLS includes a substantial variety of functionality and operating modes in consideration of the variety of applications with which TLS is used.

The desire described above led to the inclusion of a substantial number of ECC-based mechanisms. In order to encourage interoperability, a small subset of the mechanisms are identified as "recommended" - these mechanisms are capable of meeting the requirements of many applications and they should therefore be used unless an application profile of this document states otherwise, or unless in a particular environment considerations such as export regulations mandate otherwise.

The remainder of this document is organized as follows. Section 2 specifies key exchange algorithms for TLS using ECC. Section 3 specifies how client authentication is performed using ECC. Section 4 describes the TLS-specific data structures and computations involved in the operation of the ECC mechanisms. Section 5 defines cipher suites based on the ECC key exchange algorithms. Sections 6-8 discuss security

considerations, intellectual property rights, and acknowledgements

respectively. Section 9 supplies references cited elsewhere in the document, and Section 10 gives the authors' contact details.

2. Elliptic Curve Key Exchange Algorithms

This document defines six new key exchange algorithms based on ECC for use within TLS.

The table below summarizes the new key exchange algorithms.

Key Exchange Algorithm	Description	Key size limit
ECDH_ECDSA	ECDH with ECDSA signatures	None
ECDH_ECDSA_EXPORT	ECDH with ECDSA signatures	ECDH=163 bits, ECDSA=none
ECDH_RSA	ECDH with RSA signatures	None
ECDH_RSA_EXPORT	ECDH with RSA signatures	ECDH=163 bits, RSA = none
ECDH_anon	Anonymous ECDH, no signatures	None
ECDH_anon_EXPORT	Anonymous ECDH, no signatures	ECDH=163 bits

Table 2: Key exchange algorithms

Note that the key exchange algorithms marked "anon" do not provide authentication of the server or the client, and, like other "anon" TLS key exchange algorithms, may be subject to man-in-the-middle attacks. Implementations of these algorithms SHOULD provide authentication by other means.

The remainder of this section describes these key exchange algorithms in detail. For each key exchange algorithm, this involves specification of the contents of the handshake messages related to key exchange - server certificate, server key exchange, client certificate, and client key exchange - as well as specification of the computations involved in the calculation of the master secret.

2.1. ECDH_ECDSA

ECDH is used to compute the master secret. The server is authenticated via a certificate containing an ECDH public key signed with ECDSA.

Specifically this key exchange algorithm MUST proceed as follows:

- The server provides a static ECDH public key in the server certificate message using the format described in Section 4.1. The certificate is signed using ECDSA.

- The server key exchange message is not sent.

- Unless client authentication using ECDH is performed as specified in Sections 3.2 and 3.3, the client provides an ephemeral ECDH public key in the client key exchange message using the format described in Section 4.5. In this case, the client certificate and certificate verify message are not sent unless client authentication is performed using ECDSA as specified in Section 3.1, or another signature algorithm.

- If client authentication using ECDH is performed, the client provides a static ECDH public key in the client certificate message using the format described in Section 4.4. In this case an empty client key exchange message is sent using the format described in Section 4.5, and the certificate verify message is not sent.

- The client and server compute the master secret using their ECDH key pairs as specified in Section 4.7.

ECDH computations for this key exchange algorithm are performed according to IEEE 1363 [IEEE1363] - using the ECKAS-DH1 scheme with the ECSVDP-DH secret value derivation primitive, and the KDF1 key derivation primitive using SHA-1 [FIPS180-1]. ECDSA computations are performed according to ANSI X9.62 [ANSIX962] using the hash function SHA-1 [FIPS180-1].

2.2. ECDH_ECDSA_EXPORT

Export-strength ECDH is used to compute the master secret. The server is authenticated via a certificate containing an ECDH public key signed with ECDSA.

This key exchange algorithm MUST proceed in the same way as ECDH_ECDSA, except that the key size for ECDH public keys is constrained to 163 bits or less. Here the key size of an elliptic curve public key refers to the size of the underlying finite field over which the elliptic curve is defined.

2.3. ECDH_RSA

ECDH is used to compute the master secret. The server is authenticated via a certificate containing an ECDH public key signed with RSA.

This key exchange MUST proceed in the same way as ECDH_ECDSA, except that the server's certificate is signed with RSA.

2.4. ECDH_RSA_EXPORT

Export-strength ECDH is used to compute the master secret. The server is authenticated via a certificate containing an ECDH public key signed with RSA.

This key exchange algorithm MUST proceed in the same way as ECDH_RSA, except that the key size for ECDH public keys is constrained to be 163 bits or less.

2.5. ECDH_anon

Anonymous ECDH is used to compute the master secret.

Specifically this key exchange algorithm MUST proceed as follows:

- The server certificate message is not sent.

- The server provides an ephemeral ECDH public key in the server key exchange message using the format described in Section 4.2.

- The client certificate message is not sent.

- The client provides an ephemeral ECDH public key in the client key exchange message using the format described in Section 4.5.

- The client and server compute the master secret using their ECDH key pairs as specified in Section 4.7.

ECDH computations for this key exchange algorithm are performed according to IEEE 1363 [IEEE1363] - using the ECKAS-DH1 scheme with the ECSVDP-DH secret value derivation primitive, and the KDF1 key derivation primitive using SHA-1 [FIPS180-1].

2.6. ECDH_anon_EXPORT

Export-strength, anonymous ECDH is used to compute the master secret. This key exchange algorithm MUST proceed in the same way as ECDH_anon, except that the key size for ECDH public keys is constrained to 163 bits or less.

3. ECC Client Authentication

This document defines three new ECC-based client authentication methods - ECDSA_sign, ECDSA_fixed_ECDH, and RSA_fixed_ECDH.

To encourage interoperability, implementations SHOULD support ECDSA_fixed_ECDH. Implementations MAY support any of the other client authentication methods.

The remainder of this section specifies these ECC-based client authentication methods. The following information is provided for each method: which key exchange algorithms the method may be used with, what constraints apply to the method, and the formats of the certificate request, client certificate, client key exchange, and certificate verify messages.

3.1. ECDSA_sign

The client supplies a certificate containing an ECDSA public key, and authenticates itself by signing the certificate verify message with its ECDSA key pair.

This client authentication method MUST proceed as follows.

Applicable key exchange algorithms:

- This client authentication method is eligible for use with all the non-anonymous ECC-based key exchange algorithms specified in Section 2, and all the existing non-anonymous TLS key exchange algorithms specified in [TLS].

Restrictions:

- In order to perform this method, the client must possess a certified ECDSA public key.

Message exchange:

- The server requests use of the ECDSA_sign method by sending a certificate request message containing the value "ecdsa_sign" using the format described in Section 4.3. When the client receives this request, it checks that it possesses an appropriate certificate and that it is willing to proceed.

- If the client proceeds, it sends its certificate containing its ECDSA public key in the client certificate message using the format described in Section 4.4. It signs the handshake messages exchanged so far with its ECDSA key pair and conveys the resulting signature to the server in the certificate verify message using the format specified in Section 4.6.

- (If the client does not proceed, it may perform client authentication using another method suggested by the server in the certificate request message in which case the client certificate and the certificate verify message are sent in accordance with the selected method, or it may proceed with the key exchange without client authentication - in which case the client certificate and certificate verify messages are not sent.)

ECDSA computations for this client authentication method are performed according to ANSI X9.62 [ANSIX962] using the hash function SHA-1 [FIPS180-1].

3.2. ECDSA_fixed_ECDH

The client supplies an ECDSA-signed certificate containing an ECDH public key using the same elliptic curve domain parameters as the server's ECDH public key. The client authenticates itself by computing the master secret and the finished message. (This achieves authentication because these computations can only be performed by a party possessing the private key corresponding to one of the ECDH public keys exchanged.)

This client authentication method MUST proceed as follows.

Applicable key exchange algorithms:

- This method is eligible for use with all the non-anonymous ECC-based key exchange algorithms specified in Section 2.

Restrictions:

- In order to perform this client authentication method, the client must possess an ECDSA-signed certificate containing an ECDH public key using the same elliptic curve domain parameters as the ECDH public key supplied by the server in the server certificate message.

Message exchange:

- The server requests use of the ECDSA_fixed_ECDH method by sending a certificate request message containing the value "ecdsa_fixed_ecdh" using the format described in Section 4.3. When the client receives this request, it checks that it possesses an appropriate certificate and that it is willing to proceed.

- If the client proceeds, it sends its certificate containing its ECDH public key in the client certificate message using the format described in Section 4.4. It sends an empty client key exchange message using the format described in Section 4.5. It does not send the certificate verify message. It uses its static ECDH key pair, along with the server's ECDH public key) when computing the master

secret and finished message.

- (If the client does not proceed, it may perform client authentication using another method suggested by the server in the certificate request message, or it may proceed with the key exchange without client authentication - in which case the client certificate and certificate verify messages are not sent.)

ECDH computations for this key exchange algorithm are performed according to IEEE 1363 [IEEE1363] - using the ECKAS-DH1 scheme with the ECSVDP-DH secret value derivation primitive, and the KDF1 key derivation primitive using SHA-1 [FIPS180-1]. ECDSA computations are performed according to ANSI X9.62 [ANSIX962] using the hash function SHA-1 [FIPS180-1].

3.3. RSA_fixed_ECDH

The client supplies an RSA-signed certificate containing an ECDH public key using the same elliptic curve domain parameters as the server's ECDH public key. The client authenticates itself by computing the master secret and the finished message. (This achieves authentication because these computations can only be performed by a party possessing the private key corresponding to one of the ECDH public keys exchanged.)

This client authentication method MUST proceed in the same manner as the ECDSA_fixed_ECDH method, except that the client's certificate must be signed with RSA, and the server requests use of the method by sending a certificate request message containing the value "rsa_fixed_ecdh".

4. Data Structures and Computations

This section specifies the data structures and computations used by the ECC-based mechanisms specified in Sections 2 and 3. The presentation language used here is the same as that used in RFC 2246 [TLS]. 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.

4.1. Server Certificate

This message is sent in the following key exchange algorithms:

All the non-anonymous ECC-based key exchange algorithms specified in Section 2.

Meaning of this message:

This message is used to authentically convey the server's static public key to the client. The appropriate certificate types are given in the following table.

Key Exchange Algorithm	Certificate Type
ECDH_ECDSA	ECC public key; the certificate must allow the key to be used for key agreement. The certificate must be signed with ECDSA.
ECDH_ECDSA_EXPORT	ECC public key which can be used for key agreement; key size must be 163 bits or less. Certificate must be signed with ECDSA.
ECDH_RSA	ECC public key which can be used for key agreement. Certificate must be signed with RSA.
ECDH_RSA_EXPORT	ECC public key which can be used for key agreement; key size must be 163 bits or less. Certificate must be signed with RSA.

Table 3: Server certificate types

[PKIX-ALG] specifies how ECC keys and ECDSA signatures are placed in **X.509 certificates. Servers SHOULD use the elliptic curve domain** parameters recommended in ANSI X9.62 [ANSIX962], FIPS 186-2 [FIPS186-2], and SEC 2 [SEC2]. Note that - as with RSA - the same identifier is used for all ECC keys in "SubjectPublicKeyInfo". The key usage extension may be used to further delimit the use of the key. When a key usage extension is present, the "keyAgreement" bit MUST be set for ECDH certificates.

Structure of this message:

Identical to the TLS Certificate format.

Actions of the sender:

The server constructs an appropriate certificate chain and conveys it to the client in the Certificate message.

Actions of the receiver:

The client validates the certificate chain, extracts the server's public key, and checks that the key is of the correct type for the key exchange algorithm.

4.2. Server Key Exchange

This message is sent in the following key exchange algorithms:

Both the anonymous ECC-based key exchange algorithms specified in Section 2.

Meaning of this message:

This message is used to convey the server's ephemeral ECDH public key (and the corresponding elliptic curve domain parameters) to the client.

Structure of this message:

The TLS ServerKeyExchange message is extended as follows.

```
     enum { ec_diffie_hellman } KeyExchangeAlgorithm;
```

ec_diffie_hellman
Indicates the ServerKeyExchange message is to contain an ECDH public key.

```
     enum { explicit_prime (1), explicit_char2 (2),
            named_curve (3), (255) } ECCurveType;
```

explicit_prime
Indicates the elliptic curve domain parameters will be conveyed verbosely, and that the underlying finite field is a prime field.

explicit_char2
Indicates the elliptic curve domain parameters will be conveyed verbosely, and that the underlying finite field is a characteristic 2 field.

named_curve
Indicates that a named curve will be used. The use of this option is strongly recommended.

```
     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 contains the byte string representation of a field element following the conversion routine in [X9.62], section 4.3.3.

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 byte string representation of an elliptic curve point following the conversion routine in [X9.62], section 4.3.6.

```
enum { ec_basis_trinomial, ec_basis_pentanomial } ECBasisType;
```

ec_basis_trinomial
Indicates representation of a characteristic two field using a trinomial basis.

ec_basis_pentanomial
Indicates representation of a characteristic two field using a pentanomial basis.

```
enum {
    sect163k1 (1), sect163r1 (2), sect163r2 (3),
    sect193r1 (4), sect193r2 (5), sect233k1 (6),
    sect233r1 (7), sect239k1 (8), sect283k1 (9),
    sect283r1 (10), sect409k1 (11), sect409r1 (12),
    sect571k1 (13), sect571r1 (14), secp160k1 (15),
    secp160r1 (16), secp160r2 (17), secp192k1 (18),
    secp192r1 (19), secp224k1 (20), secp224r1 (21),
    secp256k1 (22), secp256r1 (23), secp384r1 (24),
    secp521r1 (25), (255)
} NamedCurve;
```

sect163k1, etc
Indicates use of the corresponding recommended curve specified in SEC 2 [SEC2]. Note that many of these curves are also recommended in ANSI X9.62 [ANSIX962], and FIPS 186-2 [FIPS186-2].

```
     struct {
         ECCurveType    curve_type;
         select (curve_type) {
             case explicit_prime:
                 opaque      prime_p <1..2^8-1>;
                 ECCurve     curve;
                 ECPoint     base;
                 opaque      order <1..2^8-1>;
                 opaque      cofactor <1..2^8-1>;
             case explicit_char2:
                 uint16      m;
                 ECBasisType basis;
                 select (basis) {
                     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>;
             case named_curve:
                 NamedCurve namedcurve;
         };
     } ECParameters;
```

curve_type
This identifies the type of the elliptic curve domain parameters.

prime_p
This is the odd prime defining the field Fp.

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

base
Specifies the base point G on the elliptic curve.

order
Specifies the order n of the base point.

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

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

k
The exponent k for the trinomial 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.

namedcurve
Specifies a recommended set of elliptic curve domain parameters.

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

curve_params
Specifies the elliptic curve domain parameters associated with the ECDH public key.

public
The ephemeral ECDH public key.

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

params
Specifies the ECDH public key and associated domain parameters.

signed_params
This element is empty for all the key exchange algorithms specified in this document.

Actions of the sender:

The server selects elliptic curve domain parameters and an ephemeral ECDH public key corresponding to these parameters according to the ECKAS-DH1 scheme from IEEE 1363 [IEEE1363]. It conveys this information to the client in the ServerKeyExchange message using the format defined above.

Actions of the recipient:

The client retrieves the server's elliptic curve domain parameters and ephemeral ECDH public key from the ServerKeyExchange message.

4.3. Certificate Request

This message is sent when requesting the following client authentication methods:

Any of the ECC-based client authentication methods specified in Section 3.

Meaning of this message:

The server uses this message to indicate which client authentication methods the server would like to use.

Structure of this message:

The TLS CertificateRequest message is extended as follows.

```
enum {
    ecdsa_sign (5), rsa_fixed_ecdh (6),
    ecdsa_fixed_ecdh(7), (255)
} ClientCertificateType;
```

ecdsa_sign, etc
Indicates that the server would like to use the corresponding client authentication method specified in Section 3.

Actions of the sender:

The server decides which client authentication methods it would like to use, and conveys this information to the client using the format defined above.

Actions of the receiver:

The client determines whether it has an appropriate certificate for use with any of the requested methods, and decides whether or not to proceed with client authentication.

4.4. Client Certificate

This message is sent in the following client authentication methods:

All the ECC-based client authentication methods specified in Section 3.

Meaning of this message:

This message is used to authentically convey the client's static public key to the server. The appropriate certificate types are given in the

following table.

Client Authentication Method	Certificate Type
ECDSA_sign	ECC public key which can be used for signing.
ECDSA_fixed_ECDH	ECC public key which can be used for key agreement. Certificate must be signed with ECDSA.
RSA_fixed_ECDH	ECC public key which can be used for key agreement. Certificate must be signed with RSA.

Table 4: Client certificate types

[PKIX-ALG] specifies how ECC keys and ECDSA signatures are placed in **X.509 certificates. Clients SHOULD use the elliptic curve domain** parameters recommended in ANSI X9.62 [ANSIX962], FIPS 186-2 [FIPS186-2], and SEC 2 [SEC2]. Note that - as with RSA - the same identifier is used for all ECC keys in "SubjectPublicKeyInfo". The key usage extension may be used to further delimit the use of the key. When a key usage extension is present, the "keyAgreement" bit MUST be set for ECDH certificates, and the "digitalSignature" bit MUST be set for ECDSA certificates.

Structure of this message:

Identical to the TLS Certificate format.

Actions of the sender:

The client constructs an appropriate certificate chain, and conveys it to the server in the Certificate message.

Actions of the receiver:

The TLS server validates the certificate chain, extracts the client's public key, and checks that the key is of the correct type for the client authentication method.

4.5. Client Key Exchange

This message is sent in the following key exchange algorithms:

All the ECC-based key exchange algorithms specified in Section 2. If client authentication with fixed ECDH is not being used, the message contains the client's ephemeral ECDH public key, otherwise the message

is empty.

Meaning of the message:

This message is used to convey ephemeral data relating to the key exchange belonging to the client (such as its ephemeral ECDH public key).

Structure of this message:

The TLS ClientKeyExchange message is extended as follows.

```
enum { yes, no } EphemeralPublicKey;
```

yes, no
Indicates whether or not the client is providing an ephemeral ECDH public key.

```
struct {
    select (EphemeralPublicKey) {
        case yes: ECPoint  ecdh_Yc;
        case no:  struct { };
    } ecdh_public;
} ClientECDiffieHellmanPublic;
```

ecdh_Yc
Contains the client's ephemeral ECDH public key.

```
struct {
    select (KeyExchangeAlgorithm) {
        case ec_diffie_hellman: ClientECDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```

Actions of the sender:

The client selects an ephemeral ECDH public key corresponding to the parameters it received from the server according to the ECKAS-DH1 scheme from IEEE 1363 [IEEE1363]. It conveys this information to the client in the ClientKeyExchange message using the format defined above.

Actions of the recipient:

The server retrieves the client's ephemeral ECDH public key from the ServerKeyExchange message and checks that the public key represents a point of the elliptic curve.

4.6. Certificate Verify

This message is sent in the following client authentication methods:

ECDSA_sign

Meaning of the message:

This message contains an ECDSA signature on the handshake messages in order to authenticate the client to the server.

Structure of this message:

The TLS CertificateVerify message is extended as follows.

```
enum { ec_dsa } SignatureAlgorithm;

select (SignatureAlgorithm) {
    case ec_dsa:
        digitally-signed struct {
            opaque sha_hash[20];
        };
} Signature;
```

In the CertificateVerify message, the signature field contains the client's ECDSA signature on the handshake messages exchanged so far. According to [ANSIX962], the signature consists of a pair of integers r and s. These integers are both converted into byte strings of the same length as the curve order n using the conversion routine specified in Section 4.3.1 of [ANSIX962], the two byte strings are concatenated, and the result is placed in the signature field.

Actions of the sender:

The client computes its signature over the handshake messages exchanged so far using its ECDSA key pair with ECDSA computations performed as specified in [ANSIX962] with the hash function SHA-1 [FIPS186-2]. The client conveys its signature to the server in the CertificateVerify message using the format defined above.

Actions of the receiver:

The server extracts the client's signature from the CertificateVerify message, and verifies the signature using the client's ECDSA public key that it received in the ClientCertificate message.

4.7. Computing the Master Secret

In all the ECC-based key exchange algorithms specified in Section 2, the client and server compute the master key as follows:

- They both compute a single shared secret K of length 20 bytes using their ECDH key pairs with ECDH computations performed as specified by the ECKAS-DH1 scheme in [IEEE1363] with the ECSVDP-DH secret value derivation primitive, and the KDF1 key derivation primitive using SHA-1 [FIPS180-1].

- They both use K as the pre_master_secret, and compute the master_secret from the pre_master_secret as specified in [TLS].

5. Cipher Suites

The table below defines the cipher suites specified in this document for use with the key exchange algorithms specified in Section 2.

```
  CipherSuite TLS_ECDH_ECDSA_WITH_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_ECDSA_WITH_AES_128_CBC_SHA       = { 0x00, 0x4B }
  CipherSuite TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA       = { 0x00, 0x4C }
  CipherSuite TLS_ECDH_ECDSA_EXPORT_WITH_RC4_40_SHA     = { 0x00, 0x4B }
  CipherSuite TLS_ECDH_ECDSA_EXPORT_WITH_RC4_56_SHA     = { 0x00, 0x4C }
  CipherSuite TLS_ECDH_RSA_WITH_NULL_SHA                = { 0x00, 0x4D }
  CipherSuite TLS_ECDH_RSA_WITH_RC4_128_SHA             = { 0x00, 0x4E }
  CipherSuite TLS_ECDH_RSA_WITH_DES_CBC_SHA             = { 0x00, 0x4F }
  CipherSuite TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA        = { 0x00, 0x50 }
  CipherSuite TLS_ECDH_RSA_WITH_AES_128_CBC_SHA         = { 0x00, 0x51 }
  CipherSuite TLS_ECDH_RSA_WITH_AES_256_CBC_SHA         = { 0x00, 0x52 }
  CipherSuite TLS_ECDH_RSA_EXPORT_WITH_RC4_40_SHA       = { 0x00, 0x53 }
  CipherSuite TLS_ECDH_RSA_EXPORT_WITH_RC4_56_SHA       = { 0x00, 0x54 }
  CipherSuite TLS_ECDH_anon_NULL_WITH_SHA               = { 0x00, 0x55 }
  CipherSuite TLS_ECDH_anon_WITH_RC4_128_SHA            = { 0x00, 0x56 }
  CipherSuite TLS_ECDH_anon_WITH_DES_CBC_SHA            = { 0x00, 0x57 }
  CipherSuite TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA       = { 0x00, 0x58 }
  CipherSuite TLS_ECDH_anon_EXPORT_WITH_DES40_CBC_SHA   = { 0x00, 0x59 }
  CipherSuite TLS_ECDH_anon_EXPORT_WITH_RC4_40_SHA      = { 0x00, 0x5A }
```

Table 5: TLS ECC cipher suites

The key exchange method, cipher, and hash algorithm for each of these cipher suites are easily determined by examining the name. Ciphers other than AES ciphers, and hash algorithms are defined in [TLS]. AES ciphers are defined in [TLS-AES].

The cipher suites which use the "NULL" cipher or one of the "EXPORT" key exchange algorithms are considered to be "exportable" cipher suites for the purposes of the TLS protocol.

Use of the following cipher suites is recommended in general - server implementations SHOULD support all of these cipher suites, and client implementations SHOULD support at least one of them:

TLS_ECDH_ECDSA_WITH_RC4_128_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA

Implementations MAY support any of the other cipher suites.

6. Security Considerations

This document is entirely concerned with security mechanisms.

This document is based on [TLS], [ANSIX9.62], and [IEEE1363] and the appropriate security considerations of those documents apply.

In addition implementers should take care to ensure that code which controls security mechanisms is free of errors which might be exploited by attackers.

7. Intellectual Property Rights

8. Acknowledgments

The authors wish to thank Bill Anderson, Paul Fahn, Gilles Garon, John Kennedy, and Brian Minard for their help preparing this document.

9. References

[ANSIX9.62] ANSI X9.62-1999, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", American National Standards Institute, 1998.

[FIPS180] FIPS 180-1, "Secure Hash Standard", National Institute of Standards and Technology, 1995.

[FIPS186-2] FIPS 186-2, "Digital Signature Standard", National Institute of Standards and Technology, 2000.

[IEEE1363] IEEE 1363, "Standard Specifications for Public Key Cryptography", Institute of Electrical and Electronics Engineers, 2000.

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

[PKIX-ALG] L. Bassham, R. Housley and W. Polk, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", PKIX Working Group Internet-Draft, draft-ietf-pkix-ipki-pkalgs-02.txt, March 2001.

[PKIX-CERT] W. Ford, R. Housley, W. Polk and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", PKIX Working Group Internet-Draft, draft-ietf-pkix-new-part1-05.txt, March 2001.

[SEC1] SEC 1, "Elliptic Curve Cryptography", Standards for Efficient Cryptography Group, 2000.

[SEC2] SEC 2, "Recommended Elliptic Curve Domain Parameters", Standards for Efficient Cryptography Group, 2000.

[TLS] T. Dierks and C. Allen, "The TLS Protocol - Version 1.0," IETF RFC 2246, January 1999.

[TLS-AES] P. Chown, "AES Ciphersuites for TLS", TLS Working Group Internet-Draft, draft-ietf-tls-ciphersuite-03.txt, January 2001.

[TLS-EXT] S. Blake-Wilson and M. Nystrom, "Wireless Extensions to TLS", TLS Working Group Internet-Draft, draft-ietf-tls-wireless-00.txt, November 2000.

10. Authors' Addresses

Authors:

Simon Blake-Wilson
Certicom Corp.
sblake-wilson@certicom.com

Tim Dierks
Certicom Corp.
timd@consensus.com

Chris Hawk
Certicom Corp.
chawk@certicom.com