Internet Engineering Task Force (IETF)
Request for Comments: 6090
Category: Informational
ISSN: 2070-1721

D. McGrew
Cisco Systems
K. Igoe
M. Salter
National Security Agency

February 2011

Fundamental Elliptic Curve Cryptography Algorithms

Abstract

This note describes the fundamental algorithms of Elliptic Curve Cryptography (ECC) as they were defined in some seminal references from 1994 and earlier. These descriptions may be useful for implementing the fundamental algorithms without using any of the specialized methods that were developed in following years. Only elliptic curves defined over fields of characteristic greater than three are in scope; these curves are those used in Suite B.

Status of This Memo

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

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6090.

Table of Contents

1. Introduction

ECC is a public-key technology that offers performance advantages at higher security levels. It includes an elliptic curve version of the Diffie-Hellman key exchange protocol [DH1976] and elliptic curve versions of the ElGamal Signature Algorithm [E1985]. The adoption of ECC has been slower than had been anticipated, perhaps due to the lack of freely available normative documents and uncertainty over intellectual property rights.

This note contains a description of the fundamental algorithms of ECC over finite fields with characteristic greater than three, based directly on original references. Its intent is to provide the Internet community with a summary of the basic algorithms that predate any specialized or optimized algorithms. The summary is detailed enough for use as a normative reference. The original descriptions and notations were followed as closely as possible.

There are several standards that specify or incorporate ECC algorithms, including the Internet Key Exchange (IKE), ANSI X9.62, and IEEE P1363. The algorithms in this note can interoperate with

some of the algorithms in these standards, with a suitable choice of parameters and options. The specifics are itemized in Section 7.

The rest of the note is organized as follows. Sections 2.1, 2.2, and 2.3 furnish the necessary terminology and notation from modular arithmetic, group theory, and the theory of finite fields, respectively. Section 3 defines the groups based on elliptic curves over finite fields of characteristic greater than three. Section 4 presents the fundamental Elliptic Curve Diffie-Hellman (ECDH) algorithm. Section 5 presents elliptic curve versions of the ElGamal signature method. The representation of integers as octet strings is specified in Section 6. Sections 2 through 6, inclusive, contain all of the normative text (the text that defines the norm for implementations conforming to this specification), and all of the following sections are purely informative. Interoperability is discussed in Section 7. Validation testing is described in Section 8. Section 9 reviews intellectual property issues. Section 10 summarizes security considerations. Appendix B describes random number generation, and other appendices provide clarifying details.

1.1. Conventions Used in This Document

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 Appendix A.

2. Mathematical Background

This section reviews mathematical preliminaries and establishes terminology and notation that are used below.

2.1. Modular Arithmetic

This section reviews modular arithmetic. Two integers x and y are said to be congruent modulo n if x - y is an integer multiple of n.

Two integers x and y are coprime when their greatest common divisor is 1; in this case, there is no third number z > 1 such that z divides x and z divides y.

The set Zq = { 0, 1, 2, ..., q-1 } is closed under the operations of modular addition, modular subtraction, modular multiplication, and modular inverse. These operations are as follows.

For each pair of integers a and b in Zq, a + b mod q is equal to a + b if a + b < q, and is equal to a + b - q otherwise.

For each pair of integers a and b in Zq, a - b mod q is equal to a - b if a - b >= 0, and is equal to a - b + q otherwise.

For each pair of integers a and b in Zq, a * b mod q is equal to the remainder of a * b divided by q.

For each integer x in Zq that is coprime with q, the inverse of x modulo q is denoted as 1/x mod q, and can be computed using the extended Euclidean algorithm (see Section 4.5.2 of [K1981v2], for example).

Algorithms for these operations are well known; for instance, see Chapter 4 of [K1981v2].

2.2. Group Operations

This section establishes some terminology and notation for mathematical groups, which are needed later on. Background references abound; see [D1966], for example.

A group is a set of elements G together with an operation that combines any two elements in G and returns a third element in G. The operation is denoted as * and its application is denoted as a * b, for any two elements a and b in G. The operation is associative, that is, for all a, b, and c in G, a * (b * c) is identical to (a * b) * c. Repeated application of the group operation N-1 times to the element a is denoted as a^N, for any element a in G and any positive integer N. That is, a^2 = a * a, a^3 = a * a * a, and so on. The associativity of the group operation ensures that the computation of a^n is unambiguous; any grouping of the terms gives the same result.

The above definition of a group operation uses multiplicative notation. Sometimes an alternative called additive notation is used, in which a * b is denoted as a + b, and a^N is denoted as N * a. In multiplicative notation, a^N is called exponentiation, while the equivalent operation in additive notation is called scalar multiplication. In this document, multiplicative notation is used throughout for consistency. Appendix E elucidates the correspondence between the two notations.

Every group has a special element called the identity element, which we denote as e. For each element a in G, e * a = a * e = a. By convention, a^0 is equal to the identity element for any a in G.

Every group element a has a unique inverse element b such that a * b = b * a = e. The inverse of a is denoted as a^-1 in multiplicative notation. (In additive notation, the inverse of a is denoted as -a.)

For any positive integer X, a^(-X) is defined to be (a^-1)^(X). Using this convention, exponentiation behaves as one would expect, namely for any integers X and Y:

a^(X+Y) = (a^X)*(a^Y)

(a^X)^Y = a^(XY) = (a^Y)^X.

In cryptographic applications, one typically deals with finite groups (groups with a finite number of elements), and for such groups, the number of elements of the group is also called the order of the group. A group element a is said to have finite order if a^X = e for some positive integer X, and the order of a is the smallest such X. If no such X exists, a is said to have infinite order. All elements of a finite group have a finite order, and the order of an element is always a divisor of the group order.

If a group element a has order R, then for any integers X and Y,

a^X = a^(X mod R),

a^X = a^Y if and only if X is congruent to Y mod R,

the set H = { a, a^2, a^3, ... , a^R=e } forms a subgroup of G, called the cyclic subgroup generated by a, and a is said to be a generator of H.

Typically, there are several group elements that generate H. Any group element of the form a^M, with M relatively prime to R, also generates H. Note that a^M is equal to g^(M modulo R) for any non-negative integer M.

Given the element a of order R, and an integer i between 1 and R-1, inclusive, the element a^i can be computed by the "square and multiply" method outlined in Section 2.1 of [M1983] (see also Knuth, Vol. 2, Section 4.6.3), or other methods.

2.3. The Finite Field Fp

This section establishes terminology and notation for finite fields with prime characteristic.

When p is a prime number, then the set Zp, with the addition, subtraction, multiplication, and division operations, is a finite field with characteristic p. Each nonzero element x in Zp has an inverse 1/x. There is a one-to-one correspondence between the integers between zero and p-1, inclusive, and the elements of the field. The field Zp is sometimes denoted as Fp or GF(p).

Equations involving field elements do not explicitly denote the "mod p" operation, but it is understood to be implicit. For example, the statement that x, y, and z are in Fp and

z = x + y

is equivalent to the statement that x, y, and z are in the set { 0, 1, ..., p-1 } and

z = x + y mod p.

3. Elliptic Curve Groups

This note only covers elliptic curves over fields with characteristic greater than three; these are the curves used in Suite B [SuiteB]. For other fields, the definition of the elliptic curve group would be different.

An elliptic curve over a field Fp is defined by the curve equation

y^2 = x^3 + a*x + b,

where x, y, a, and b are elements of the field Fp [M1985], and the discriminant is nonzero (as described in Section 3.3.1). A point on an elliptic curve is a pair (x,y) of values in Fp that satisfies the curve equation, or it is a special point (@,@) that represents the identity element (which is called the "point at infinity"). The order of an elliptic curve group is the number of distinct points.

Two elliptic curve points (x1,y1) and (x2,y2) are equal whenever x1=x2 and y1=y2, or when both points are the point at infinity. The inverse of the point (x1,y1) is the point (x1,-y1). The point at infinity is its own inverse.

The group operation associated with the elliptic curve group is as follows [BC1989]. To an arbitrary pair of points P and Q specified by their coordinates (x1,y1) and (x2,y2), respectively, the group operation assigns a third point P*Q with the coordinates (x3,y3). These coordinates are computed as follows:

(x3,y3) = (@,@) when P is not equal to Q and x1 is equal to x2.

x3 = ((y2-y1)/(x2-x1))^2 - x1 - x2 and
y3 = (x1-x3)*(y2-y1)/(x2-x1) - y1 when P is not equal to Q and x1 is not equal to x2.

(x3,y3) = (@,@) when P is equal to Q and y1 is equal to 0.

x3 = ((3*x1^2 + a)/(2*y1))^2 - 2*x1 and
y3 = (x1-x3)*(3*x1^2 + a)/(2*y1) - y1 if P is equal to Q and y1 is not equal to 0.

In the above equations, a, x1, x2, x3, y1, y2, and y3 are elements of the field Fp; thus, computation of x3 and y3 in practice must reduce the right-hand-side modulo p. Pseudocode for the group operation is provided in Appendix F.1.

The representation of elliptic curve points as a pair of integers in Zp is known as the affine coordinate representation. This representation is suitable as an external data representation for communicating or storing group elements, though the point at infinity must be treated as a special case.

Some pairs of integers are not valid elliptic curve points. A valid pair will satisfy the curve equation, while an invalid pair will not.

3.1. Homogeneous Coordinates

An alternative way to implement the group operation is to use homogeneous coordinates [K1987] (see also [KMOV1991]). This method is typically more efficient because it does not require a modular inversion operation.

An elliptic curve point (x,y) (other than the point at infinity (@,@)) is equivalent to a point (X,Y,Z) in homogeneous coordinates whenever x=X/Z mod p and y=Y/Z mod p.

Let P1=(X1,Y1,Z1) and P2=(X2,Y2,Z2) be points on an elliptic curve, and suppose that the points P1 and P2 are not equal to (@,@), P1 is not equal to P2, and P1 is not equal to P2^-1. Then the product P3=(X3,Y3,Z3) = P1 * P2 is given by

X3 = v * (Z2 * (Z1 * u^2 - 2 * X1 * v^2) - v^3) mod p

Y3 = Z2 * (3 * X1 * u * v^2 - Y1 * v^3 - Z1 * u^3) + u * v^3 mod p

Z3 = v^3 * Z1 * Z2 mod p

where u = Y2 * Z1 - Y1 * Z2 mod p and v = X2 * Z1 - X1 * Z2 mod p.

When the points P1 and P2 are equal, then (X1/Z1, Y1/Z1) is equal to (X2/Z2, Y2/Z2), which is true if and only if u and v are both equal to zero.

The product P3=(X3,Y3,Z3) = P1 * P1 is given by

X3 = 2 * Y1 * Z1 * (w^2 - 8 * X1 * Y1^2 * Z1) mod p

Y3 = 4 * Y1^2 * Z1 * (3 * w * X1 - 2 * Y1^2 * Z1) - w^3 mod p

Z3 = 8 * (Y1 * Z1)^3 mod p

where w = 3 * X1^2 + a * Z1^2 mod p. In the above equations, a, u, v, w, X1, X2, X3, Y1, Y2, Y3, Z1, Z2, and Z3 are integers in the set Fp. Pseudocode for the group operation in homogeneous coordinates is provided in Appendix F.2.

When converting from affine coordinates to homogeneous coordinates, it is convenient to set Z to 1. When converting from homogeneous coordinates to affine coordinates, it is necessary to perform a modular inverse to find 1/Z mod p.

3.2. Other Coordinates

Some other coordinate systems have been described; several are documented in [CC1986], including Jacobi coordinates.

3.3. ECC Parameters

In cryptographic contexts, an elliptic curve parameter set consists of a cyclic subgroup of an elliptic curve together with a preferred generator of that subgroup. When working over a prime order finite field with characteristic greater than three, an elliptic curve group is completely specified by the following parameters:

The prime number p that indicates the order of the field Fp.

The value a used in the curve equation.

The value b used in the curve equation.

The generator g of the subgroup.

The order n of the subgroup generated by g.

An example of an ECC parameter set is provided in Appendix D.

Parameter generation is out of scope for this note.

Each elliptic curve point is associated with a particular parameter set. The elliptic curve group operation is only defined between two points in the same group. It is an error to apply the group

operation to two elements that are from different groups, or to apply the group operation to a pair of coordinates that is not a valid point. (A pair (x,y) of coordinates in Fp is a valid point only when it satisfies the curve equation.) See Section 10.3 for further information.

3.3.1. Discriminant

For each elliptic curve group, the discriminant -16*(4*a^3 + 27*b^2) must be nonzero modulo p [S1986]; this requires that

$$4*a^3 + 27*b^2 \neq 0 \bmod p.$$

3.3.2. Security

Security is highly dependent on the choice of these parameters. This section gives normative guidance on acceptable choices. See also Section 10 for informative guidance.

The order of the group generated by g MUST be divisible by a large prime, in order to preclude easy solutions of the discrete logarithm problem [K1987].

With some parameter choices, the discrete log problem is significantly easier to solve. This includes parameter sets in which b = 0 and p = 3 (mod 4), and parameter sets in which a = 0 and p = 2 (mod 3) [MOV1993]. These parameter choices are inferior for cryptographic purposes and SHOULD NOT be used.

4. Elliptic Curve Diffie-Hellman (ECDH)

The Diffie-Hellman (DH) key exchange protocol [DH1976] allows two parties communicating over an insecure channel to agree on a secret key. It was originally defined in terms of operations in the multiplicative group of a field with a large prime characteristic. Massey [M1983] observed that it can be easily generalized so that it is defined in terms of an arbitrary cyclic group. Miller [M1985] and Koblitz [K1987] analyzed the DH protocol over an elliptic curve group. We describe DH following the former reference.

Let G be a group, and g be a generator for that group, and let t denote the order of G. The DH protocol runs as follows. Party A chooses an exponent j between 1 and t-1, inclusive, uniformly at random, computes g^j, and sends that element to B. Party B chooses an exponent k between 1 and t-1, inclusive, uniformly at random, computes g^k, and sends that element to A. Each party can compute g^(j*k); party A computes (g^k)^j, and party B computes (g^j)^k.

See Appendix B regarding generation of random integers.

4.1. Data Types

Each run of the ECDH protocol is associated with a particular parameter set (as defined in Section 3.3), and the public keys g^j and g^k and the shared secret g^(j*k) are elements of the cyclic subgroup associated with the parameter set.

An ECDH private key z is an integer in Zt, where t is the order of the subgroup.

4.2. Compact Representation

As described in the final paragraph of [M1985], the x-coordinate of the shared secret value g^(j*k) is a suitable representative for the entire point whenever exponentiation is used as a one-way function. In the ECDH key exchange protocol, after the element g^(j*k) has been computed, the x-coordinate of that value can be used as the shared secret. We call this compact output.

Following [M1985] again, when compact output is used in ECDH, only the x-coordinate of an elliptic curve point needs to be transmitted, instead of both coordinates as in the typical affine coordinate representation. We call this the compact representation. Its mathematical background is explained in Appendix C.

ECDH can be used with or without compact output. Both parties in a particular run of the ECDH protocol MUST use the same method. ECDH can be used with or without compact representation. If compact representation is used in a particular run of the ECDH protocol, then compact output MUST be used as well.

5. Elliptic Curve ElGamal Signatures

5.1. Background

The ElGamal signature algorithm was introduced in 1984 [E1984a] [E1984b] [E1985]. It is based on the discrete logarithm problem, and was originally defined for the multiplicative group of the integers modulo a large prime number. It is straightforward to extend it to use other finite groups, such as the multiplicative group of the finite field GF(2^w) [AMV1990] or an elliptic curve group [A1992].

An ElGamal signature consists of a pair of components. There are many possible generalizations of ElGamal signature methods that have been obtained by different rearrangements of the equation for the second component; see [HMP1994], [HP1994], [NR1994], [A1992], and

[AMV1990]. These generalizations are independent of the mathematical group used, and have been described for the multiplicative group modulo a prime number, the multiplicative group of GF(2^w), and elliptic curve groups [HMP1994] [NR1994] [AMV1990] [A1992].

The Digital Signature Algorithm (DSA) [FIPS186] is an important ElGamal signature variant.

5.2. Hash Functions

ElGamal signatures must use a collision-resistant hash function, so that it can sign messages of arbitrary length and can avoid existential forgery attacks; see Section 10.4. (This is true for all ElGamal variants [HMP1994].) We denote the hash function as h(). Its input is a bit string of arbitrary length, and its output is a non-negative integer.

Let H() denote a hash function whose output is a fixed-length bit string. To use H in an ElGamal signature method, we define the mapping between that output and the non-negative integers; this realizes the function h() described above. Given a bit string m, the function h(m) is computed as follows:

1. H(m) is evaluated; the result is a fixed-length bit string.

2. Convert the resulting bit string to an integer i by treating its leftmost (initial) bit as the most significant bit of i, and treating its rightmost (final) bit as the least significant bit of i.

5.3. KT-IV Signatures

Koyama and Tsuruoka described a signature method based on Elliptic Curve ElGamal, in which the first signature component is the x-coordinate of an elliptic curve point reduced modulo q [KT1994]. In this section, we recall that method, which we refer to as KT-IV.

The algorithm uses an elliptic curve group, as described in Section 3.3, with prime field order p and curve equation parameters a and b. We denote the generator as alpha, and the order of the generator as q. We follow [FIPS186] in checking for exceptional cases.

5.3.1. Keypair Generation

The private key z is an integer between 1 and q-1, inclusive, generated uniformly at random. (See Appendix B regarding random integers.) The public key is the group element Y = alpha^z. Each

public key is associated with a particular parameter set as per Section 3.3.

5.3.2. Signature Creation

To compute a KT-IV signature for a message m using the private key z:

1. Choose an integer k uniformly at random from the set of all integers between 1 and q-1, inclusive. (See Appendix B regarding random integers.)

2. Calculate R = (r_x, r_y) = alpha^k.

3. Calculate s1 = r_x mod q.

4. Check if h(m) + z * s1 = 0 mod q; if so, a new value of k MUST be generated and the signature MUST be recalculated. As an option, one MAY check if s1 = 0; if so, a new value of k SHOULD be generated and the signature SHOULD be recalculated. (It is extremely unlikely that s1 = 0 or h(m) + z * s1 = 0 mod q if signatures are generated properly.)

5. Calculate s2 = k/(h(m) + z*s1) mod q.

The signature is the ordered pair (s1, s2). Both signature components are non-negative integers.

5.3.3. Signature Verification

Given the message m, the generator g, the group order q, the public key Y, and the signature (s1, s2), verification is as follows:

1. Check to see that 0 < s1 < q and 0 < s2 < q; if either condition is violated, the signature SHALL be rejected.

2. Compute the non-negative integers u1 and u2, where

   u1 = h(m) * s2 mod q, and

   u2 = s1 * s2 mod q.

3. Compute the elliptic curve point R' = alpha^u1 * Y^u2.

4. If the x-coordinate of R' mod q is equal to s1, then the signature and message pass the verification; otherwise, they fail.

5.4. KT-I Signatures

Horster, Michels, and Petersen categorized many different ElGamal signature methods, demonstrated their equivalence, and showed how to convert signatures of one type to another type [HMP1994]. In their terminology, the signature method of Section 5.3 and [KT1994] is a Type IV method, which is why it is denoted as KT-IV.

A Type I KT signature method has a second component that is computed in the same manner as that of the Digital Signature Algorithm. In this section, we describe this method, which we refer to as KT-I.

5.4.1. Keypair Generation

Keypairs and keypair generation are exactly as in Section 5.3.1.

5.4.2. Signature Creation

To compute a KT-I signature for a message m using the private key z:

1. Choose an integer k uniformly at random from the set of all integers between 1 and q-1, inclusive. (See Appendix B regarding random integers.)

2. Calculate R = (r_x, r_y) = alpha^k.

3. Calculate s1 = r_x mod q.

4. Calculate s2 = (h(m) + z*s1)/k mod q.

5. As an option, one MAY check if s1 = 0 or s2 = 0. If either s1 = 0 or s2 = 0, a new value of k SHOULD be generated and the signature SHOULD be recalculated. (It is extremely unlikely that s1 = 0 or s2 = 0 if signatures are generated properly.)

The signature is the ordered pair (s1, s2). Both signature components are non-negative integers.

5.4.3. Signature Verification

Given the message m, the public key Y, and the signature (s1, s2), verification is as follows:

1. Check to see that 0 < s1 < q and 0 < s2 < q; if either condition is violated, the signature SHALL be rejected.

2. Compute s2_inv = 1/s2 mod q.

3. Compute the non-negative integers u1 and u2, where

   u1 = h(m) * s2_inv mod q, and

   u2 = s1 * s2_inv mod q.

4. Compute the elliptic curve point R' = alpha^u1 * Y^u2.

5. If the x-coordinate of R' mod q is equal to s1, then the signature and message pass the verification; otherwise, they fail.

5.5. Converting KT-IV Signatures to KT-I Signatures

A KT-IV signature for a message m and a public key Y can easily be converted into a KT-I signature for the same message and public key. If (s1, s2) is a KT-IV signature for a message m, then (s1, 1/s2 mod q) is a KT-I signature for the same message [HMP1994].

The conversion operation uses only public information, and it can be performed by the creator of the pre-conversion KT-IV signature, the verifier of the post-conversion KT-I signature, or by any other entity.

An implementation MAY use this method to compute KT-I signatures.

5.6. Rationale

This subsection is not normative for this specification and is provided only as background information.

[HMP1994] presents many generalizations of ElGamal signatures. Equation (5) of that reference shows the general signature equation

   A = x_A * B + k * C (mod q)

where x_A is the private key, k is the secret value, and A, B, and C are determined by the Type of the equation, as shown in Table 1 of [HMP1994]. DSA [FIPS186] is an EG-I.1 signature method (as is KT-I), with A = m, B = -r, and C = s. (Here we use the notation of [HMP1994] in which the first signature component is r and the second signature component is s; in KT-I and KT-IV these components are denoted as s1 and s2, respectively. The private key x_A corresponds to the private key z.) Its signature equation is

   m = -r * z + s * k (mod q).

The signature method of [KT1994] and Section 5.3 is an EG-IV.1 method, with A = m * s, B = -r * s, C = 1. Its signature equation is

```
m * s = -r * s * z + k (mod q)
```

The functions f and g mentioned in Table 1 of [HMP1994] are merely multiplication, as described under the heading "Fifth generalization".

In the above equations, we rely on the implicit conversion of the message m from a bit string to an integer. No hash function is shown in these equations, but, as described in Section 10.4, a hash function should be applied to the message prior to signing in order to prevent existential forgery attacks.

Nyberg and Rueppel [NR1994] studied many different ElGamal signature methods and defined "strong equivalence" as follows:

> Two signature methods are called strongly equivalent if the signature of the first scheme can be transformed efficiently into signatures of the second scheme and vice versa, without knowledge of the private key.

KT-I and KT-IV signatures are obviously strongly equivalent.

A valid signature with s2=0 leaks the secret key, since in that case z = -h(m) / s1 mod q. We follow [FIPS186] in checking for this exceptional case and the case that s1=0. The s2=0 check was suggested by Rivest [R1992] and is discussed in [BS1992].

[KT1994] uses "a positive integer q' that does not exceed q" when computing the signature component s1 from the x-coordinate r_x of the elliptic curve point R = (r_x, r_y). The value q' is also used during signature validation when comparing the x-coordinate of a computed elliptic curve point to the value to s1. In this note, we use the simplifying convention that q' = q.

6. Converting between Integers and Octet Strings

A method for the conversion between integers and octet strings is specified in this section, following the established conventions of public key cryptography [R1993]. This method allows integers to be represented as octet strings that are suitable for transmission or storage. This method SHOULD be used when representing an elliptic curve point or an elliptic curve coordinate as they are defined in this note.

6.1. Octet-String-to-Integer Conversion

The octet string S shall be converted to an integer x as follows. Let S1, ..., Sk be the octets of S from first to last. Then the integer x shall satisfy

$$x = \sum_{i=1}^{k} 2^{(8(k-i))} Si \ .$$

In other words, the first octet of S has the most significance in the integer and the last octet of S has the least significance.

Note: the integer x satisfies 0 <= x < 2^(8*k).

6.2. Integer-to-Octet-String Conversion

The integer x shall be converted to an octet string S of length k as follows. The string S shall satisfy

$$y = \sum_{i=1}^{k} 2^{(8(k-i))} Si \ .$$

where S1, ..., Sk are the octets of S from first to last.

In other words, the first octet of S has the most significance in the integer, and the last octet of S has the least significance.

7. Interoperability

The algorithms in this note can be used to interoperate with some other ECC specifications. This section provides details for each algorithm.

7.1. ECDH

Section 4 can be used with the Internet Key Exchange (IKE) versions one [RFC2409] or two [RFC5996]. These algorithms are compatible with the ECP groups defined by [RFC5903], [RFC5114], [RFC2409], and [RFC2412]. The group definition in this protocol uses an affine coordinate representation of the public key. [RFC5903] uses the compact output of Section 4.2, while [RFC4753] (which was obsoleted by RFC 5903) does not. Neither of those RFCs use compact representation. Note that some groups indicate that the curve parameter "a" is negative; these values are to be interpreted modulo the order of the field. For example, a parameter of a = -3 is equal to p - 3, where p is the order of the field. The test cases in

Section 8 of [RFC5903] can be used to test an implementation; these cases use the multiplicative notation, as does this note. The KEi and KEr payloads are equal to g^j and g^k, respectively, with 64 bits of encoding data prepended to them.

The algorithms in Section 4 can be used to interoperate with the IEEE [P1363] and ANSI [X9.62] standards for ECDH based on fields of characteristic greater than three. IEEE P1363 ECDH can be used in a manner that will interoperate with this note, with the following options and parameter choices from that specification:

prime curves with a cofactor of 1,

the ECSVDP-DH (Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman version),

the Key Derivation Function (KDF) must be the "identity" function (equivalently, the KDF step should be omitted and the shared secret value should be output directly).

7.2. KT-I and ECDSA

The Digital Signature Algorithm (DSA) is based on the discrete logarithm problem over the multiplicative subgroup of the finite field with large prime order [DSA1991] [FIPS186]. The Elliptic Curve Digital Signature Algorithm (ECDSA) [P1363] [X9.62] is an elliptic curve version of DSA.

KT-I is mathematically and functionally equivalent to ECDSA, and can interoperate with the IEEE [P1363] and ANSI [X9.62] standards for Elliptic Curve DSA (ECDSA) based on fields of characteristic greater than three. KT-I signatures can be verified using the ECDSA verification algorithm, and ECDSA signatures can be verified using the KT-I verification algorithm.

8. Validating an Implementation

It is essential to validate the implementation of a cryptographic algorithm. This section outlines tests that should be performed on the algorithms defined in this note.

A known answer test, or KAT, uses a fixed set of inputs to test an algorithm; the output of the algorithm is compared with the expected output, which is also a fixed value. KATs for ECDH and KT-I are set out in the following subsections.

A consistency test generates inputs for one algorithm being tested using a second algorithm that is also being tested, then checks the output of the first algorithm. A signature creation algorithm can be tested for consistency against a signature verification algorithm. Implementations of KT-I should be tested in this way. Their signature generation processes are non-deterministic, and thus cannot be tested using a KAT. Signature verification algorithms, on the other hand, are deterministic and should be tested via a KAT. This combination of tests provides coverage for all of the operations, including keypair generation. Consistency testing should also be applied to ECDH.

8.1. ECDH

An ECDH implementation can be validated using the known answer test cases from [RFC5903] or [RFC5114]. The correspondence between the notation in RFC 5903 and the notation in this note is summarized in the following table. (Refer to Sections 3.3 and 4; the generator g is expressed in affine coordinate representation as (gx, gy)).

ECDH	RFC 5903
order p of field Fp	p
curve coefficient a	-3
curve coefficient b	b
generator g	g=(gx, gy)
private keys j and k	i and r
public keys g^j, g^k	g^i = (gix, giy) and g^r = (grx, gry)

The correspondence between the notation in RFC 5114 and the notation in this note is summarized in the following table.

ECDH	RFC 5114
order p of field Fp	p
curve coefficient a	a
curve coefficient b	b
generator g	g=(gx, gy)
group order n	n
private keys j and k	dA and dB
public keys g^j, g^k	g^(dA) = (x_qA, y_qA) and g^(dB) = (x_qB, y_qB)
shared secret g^(j*k)	g^(dA*dB) = (x_Z, y_Z)

8.2. KT-I

A KT-I implementation can be validated using the known answer test cases from [RFC4754]. The correspondence between the notation in that RFC and the notation in this note is summarized in the following table.

KT-I	RFC 4754
order p of field Fp	p
curve coefficient a	-3
curve coefficient b	b
generator alpha	g
group order q	q
private key z	w
public key Y	g^w = (gwx,gwy)
random k	ephem priv k
s1	r
s2	s
s2_inv	sinv
u1	u = h*sinv mod q
u2	v = r*sinv mod q

9. Intellectual Property

Concerns about intellectual property have slowed the adoption of ECC because a number of optimizations and specialized algorithms have been patented in recent years.

All of the normative references for ECDH (as defined in Section 4) were published during or before 1989, and those for KT-I were published during or before May 1994. All of the normative text for these algorithms is based solely on their respective references.

9.1. Disclaimer

This document is not intended as legal advice. Readers are advised to consult their own legal advisers if they would like a legal interpretation of their rights.

The IETF policies and processes regarding intellectual property and patents are outlined in [RFC3979] and [RFC4879] and at https://datatracker.ietf.org/ipr/about/.

10. Security Considerations

The security level of an elliptic curve cryptosystem is determined by the cryptanalytic algorithm that is the least expensive for an attacker to implement. There are several algorithms to consider.

The Pohlig-Hellman method is a divide-and-conquer technique [PH1978]. If the group order n can be factored as

```
n = q1 * q2 * ... * qz,
```

then the discrete log problem over the group can be solved by independently solving a discrete log problem in groups of order q1, q2, ..., qz, then combining the results using the Chinese remainder theorem. The overall computational cost is dominated by that of the discrete log problem in the subgroup with the largest order.

Shanks' algorithm [K1981v3] computes a discrete logarithm in a group of order n using O(sqrt(n)) operations and O(sqrt(n)) storage. The Pollard rho algorithm [P1978] computes a discrete logarithm in a group of order n using O(sqrt(n)) operations, with a negligible amount of storage, and can be efficiently parallelized [VW1994].

The Pollard lambda algorithm [P1978] can solve the discrete logarithm problem using O(sqrt(w)) operations and O(log(w)) storage, when the exponent is known to lie in an interval of width w.

The algorithms described above work in any group. There are specialized algorithms that specifically target elliptic curve groups. There are no known subexponential algorithms against general elliptic curve groups, though there are methods that target certain special elliptic curve groups; see [MOV1993] and [FR1994].

10.1. Subgroups

A group consisting of a nonempty set of elements S with associated group operation * is a subgroup of the group with the set of elements G, if the latter group uses the same group operation and S is a subset of G. For each elliptic curve equation, there is an elliptic curve group whose group order is equal to the order of the elliptic curve; that is, there is a group that contains every point on the curve.

The order m of the elliptic curve is divisible by the order n of the group associated with the generator; that is, for each elliptic curve group, m = n * c for some number c. The number c is called the "cofactor" [P1363]. Each ECC parameter set as in Section 3.3 is associated with a particular cofactor.

It is possible and desirable to use a cofactor equal to 1.

10.2. Diffie-Hellman

Note that the key exchange protocol as defined in Section 4 does not protect against active attacks; Party A must use some method to ensure that (g^k) originated with the intended communicant B, rather than an attacker, and Party B must do the same with (g^j).

It is not sufficient to authenticate the shared secret g^(j*k), since this leaves the protocol open to attacks that manipulate the public keys. Instead, the values of the public keys g^x and g^y that are exchanged should be directly authenticated. This is the strategy used by protocols that build on Diffie-Hellman and that use end-entity authentication to protect against active attacks, such as OAKLEY [RFC2412] and the Internet Key Exchange [RFC2409] [RFC4306] [RFC5996].

When the cofactor of a group is not equal to 1, there are a number of attacks that are possible against ECDH. See [VW1996], [AV1996], and [LL1997].

10.3. Group Representation and Security

The elliptic curve group operation does not explicitly incorporate the parameter b from the curve equation. This opens the possibility that a malicious attacker could learn information about an ECDH private key by submitting a bogus public key [BMM2000]. An attacker can craft an elliptic curve group G' that has identical parameters to a group G that is being used in an ECDH protocol, except that b is different. An attacker can submit a point on G' into a run of the ECDH protocol that is using group G, and gain information from the fact that the group operations using the private key of the device under attack are effectively taking place in G' instead of G.

This attack can gain useful information about an ECDH private key that is associated with a static public key, i.e., a public key that is used in more than one run of the protocol. However, it does not gain any useful information against ephemeral keys.

This sort of attack is thwarted if an ECDH implementation does not assume that each pair of coordinates in Zp is actually a point on the appropriate elliptic curve.

These considerations also apply when ECDH is used with compact representation (see Appendix C).

10.4. Signatures

Elliptic curve parameters should only be used if they come from a trusted source; otherwise, some attacks are possible [AV1996] [V1996].

If no hash function is used in an ElGamal signature system, then the system is vulnerable to existential forgeries, in which an attacker who does not know a private key can generate valid signatures for the associated public key, but cannot generate a signature for a message of its own choosing. (See [E1985] for instance.) The use of a collision-resistant hash function eliminates this vulnerability.

In principle, any collision-resistant hash function is suitable for use in KT signatures. To facilitate interoperability, we recognize the following hashes as suitable for use as the function H defined in Section 5.2:

SHA-256, which has a 256-bit output.

SHA-384, which has a 384-bit output.

SHA-512, which has a 512-bit output.

All of these hash functions are defined in [FIPS180-2].

The number of bits in the output of the hash used in KT signatures should be equal or close to the number of bits needed to represent the group order.

11. Acknowledgements

The author expresses his thanks to the originators of elliptic curve cryptography, whose work made this note possible, and all of the reviewers, who provided valuable constructive feedback. Thanks are especially due to Howard Pinder, Andrey Jivsov, Alfred Hoenes (who contributed the algorithms in Appendix F), Dan Harkins, and Tina Tsou.

12. References

12.1. Normative References

[AMV1990] Agnew, G., Mullin, R., and S. Vanstone, "Improved Digital Signature Scheme based on Discrete Exponentiation", Electronics Letters Vol. 26, No. 14, July, 1990.

[BC1989] Bender, A. and G. Castagnoli, "On the Implementation of Elliptic Curve Cryptosystems", Advances in Cryptology - CRYPTO '89 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 435, 1989.

[CC1986] Chudnovsky, D. and G. Chudnovsky, "Sequences of numbers generated by addition in formal groups and new primality and factorization tests", Advances in Applied Mathematics, Volume 7, Issue 4, December 1986.

[D1966] Deskins, W., "Abstract Algebra", MacMillan Company New York, 1966.

[DH1976] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions in Information Theory IT-22, pp. 644-654, 1976.

[FR1994] Frey, G. and H. Ruck, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves.", Mathematics of Computation Vol. 62, No. 206, pp. 865-874, 1994.

[HMP1994] Horster, P., Michels, M., and H. Petersen, "Meta-ElGamal signature schemes", University of Technology Chemnitz-Zwickau Department of Computer Science, Technical Report TR-94-5, May 1994.

[K1981v2] Knuth, D., "The Art of Computer Programming, Vol. 2: Seminumerical Algorithms", Addison Wesley , 1981.

[K1987] Koblitz, N., "Elliptic Curve Cryptosystems", Mathematics of Computation, Vol. 48, 1987, pp. 203-209, 1987.

[KT1994] Koyama, K. and Y. Tsuruoka, "Digital signature system based on elliptic curve and signer device and verifier device for said system", Japanese Unexamined Patent Application Publication H6-43809, February 18, 1994.

[M1983] Massey, J., "Logarithms in finite cyclic groups - cryptographic issues", Proceedings of the 4th Symposium on Information Theory, 1983.

[M1985] Miller, V., "Use of elliptic curves in cryptography", Advances in Cryptology - CRYPTO '85 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 218, 1985.

[MOV1993] Menezes, A., Vanstone, S., and T. Okamoto, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field", IEEE Transactions on Information Theory Vol. 39, No. 5, pp. 1639-1646, September, 1993.

[R1993] RSA Laboratories, "PKCS#1: RSA Encryption Standard", Technical Note version 1.5, 1993.

[S1986] Silverman, J., "The Arithmetic of Elliptic Curves", Springer-Verlag, New York, 1986.

12.2. Informative References

[A1992] Anderson, J., "Response to the proposed DSS", Communications of the ACM, v. 35, n. 7, p. 50-52, July 1992.

[AV1996] Anderson, R. and S. Vaudenay, "Minding Your P's and Q's", Advances in Cryptology - ASIACRYPT '96 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 1163, 1996.

[BMM2000] Biehl, I., Meyer, B., and V. Muller, "Differential fault analysis on elliptic curve cryptosystems", Advances in Cryptology - CRYPTO 2000 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 1880, 2000.

[BS1992] Branstad, D. and M. Smid, "Response to Comments on the NIST Proposed Digital Signature Standard", Advances in Cryptology - CRYPTO '92 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 740, August 1992.

[DSA1991] U.S. National Institute of Standards and Technology, "DIGITAL SIGNATURE STANDARD", Federal Register, Vol. 56, August 1991.

[E1984a] ElGamal, T., "Cryptography and logarithms over finite fields", Stanford University, UMI Order No. DA 8420519, 1984.

[E1984b] ElGamal, T., "Cryptography and logarithms over finite fields", Advances in Cryptology - CRYPTO '84 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 196, 1984.

[E1985] ElGamal, T., "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Transactions on Information Theory, Vol. 30, No. 4, pp. 469-472, 1985.

[FIPS180-2] U.S. National Institute of Standards and Technology, "SECURE HASH STANDARD", Federal Information Processing Standard (FIPS) 180-2, August 2002.

[FIPS186] U.S. National Institute of Standards and Technology, "DIGITAL SIGNATURE STANDARD", Federal Information Processing Standard FIPS-186, May 1994.

[HP1994] Horster, P. and H. Petersen, "Verallgemeinerte ElGamal-Signaturen", Proceedings der Fachtagung SIS '94, Verlag der Fachvereine, Zurich, 1994.

[K1981v3] Knuth, D., "The Art of Computer Programming, Vol. 3: Sorting and Searching", Addison Wesley, 1981.

[KMOV1991] Koyama, K., Maurer, U., Vanstone, S., and T. Okamoto, "New Public-Key Schemes Based on Elliptic Curves over the Ring Zn", Advances in Cryptology - CRYPTO '91 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 576, 1991.

[L1969] Lehmer, D., "Computer technology applied to the theory of numbers", M.A.A. Studies in Mathematics, 180-2, 1969.

[LL1997] Lim, C. and P. Lee, "A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup", Advances in Cryptology - CRYPTO '97 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 1294, 1997.

[NR1994] Nyberg, K. and R. Rueppel, "Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem", Advances in Cryptology - EUROCRYPT '94 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 950, May 1994.

[P1363] "Standard Specifications for Public Key Cryptography", Institute of Electric and Electronic Engineers (IEEE), P1363, 2000.

[P1978] Pollard, J., "Monte Carlo methods for index computation mod p", Mathematics of Computation, Vol. 32, 1978.

[PH1978] Pohlig, S. and M. Hellman, "An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance", IEEE Transactions on Information Theory, Vol. 24, pp. 106-110, 1978.

[R1988] Rose, H., "A Course in Number Theory", Oxford University Press, 1988.

[R1992] Rivest, R., "Response to the proposed DSS", Communications of the ACM, v. 35, n. 7, p. 41-47, July 1992.

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

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC2412] Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[RFC3979] Bradner, S., "Intellectual Property Rights in IETF Technology", BCP 79, RFC 3979, March 2005.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4753] Fu, D. and J. Solinas, "ECP Groups For IKE and IKEv2", RFC 4753, January 2007.

[RFC4754] Fu, D. and J. Solinas, "IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 4754, January 2007.

[RFC4879] Narten, T., "Clarification of the Third Party Disclosure Procedure in RFC 3979", BCP 79, RFC 4879, April 2007.

[RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, January 2008.

[RFC5903] Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2", RFC 5903, June 2010.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

[SuiteB] U. S. National Security Agency (NSA), "NSA Suite B Cryptography", <http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml>.

[V1996] Vaudenay, S., "Hidden Collisions on DSS", Advances in Cryptology - CRYPTO '96 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 1109, 1996.

[VW1994] van Oorschot, P. and M. Wiener, "Parallel Collision Search with Application to Hash Functions and Discrete Logarithms", Proceedings of the 2nd ACM Conference on Computer and communications security, pp. 210-218, 1994.

[VW1996] van Oorschot, P. and M. Wiener, "On Diffie-Hellman key agreement with short exponents", Advances in Cryptology - EUROCRYPT '96 Proceedings, Springer Lecture Notes in Computer Science (LNCS), volume 1070, 1996.

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

Appendix A. Key Words

The definitions of these key words are quoted from [RFC2119] and are commonly used in Internet standards. They are reproduced in this note in order to avoid a normative reference from after 1994.

1. MUST - This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.

2. MUST NOT - This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.

3. SHOULD - This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

4. SHOULD NOT - This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. MAY - This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

Appendix B. Random Integer Generation

It is easy to generate an integer uniformly at random between zero and (2^t)-1, inclusive, for some positive integer t. Generate a random bit string that contains exactly t bits, and then convert the bit string to a non-negative integer by treating the bits as the coefficients in a base-2 expansion of an integer.

It is sometimes necessary to generate an integer r uniformly at random so that r satisfies a certain property P, for example, lying within a certain interval. A simple way to do this is with the rejection method:

1. Generate a candidate number c uniformly at random from a set that includes all numbers that satisfy property P (plus some other numbers, preferably not too many)

2. If c satisfies property P, then return c. Otherwise, return to Step 1.

For example, to generate a number between 1 and n-1, inclusive, repeatedly generate integers between zero and (2^t)-1, inclusive, stopping at the first integer that falls within that interval.

Recommendations on how to generate random bit strings are provided in [RFC4086].

Appendix C. Why Compact Representation Works

In the affine representation, the x-coordinate of the point P^i does not depend on the y-coordinate of the point P, for any non-negative exponent i and any point P. This fact can be seen as follows. When given only the x-coordinate of a point P, it is not possible to determine exactly what the y-coordinate is, but the y value will be a solution to the curve equation

```
y^2 = x^3 + a*x + b (mod p).
```

There are at most two distinct solutions y = w and y = -w mod p, and the point P must be either Q=(x,w) or Q^-1=(x,-w). Thus P^n is equal to either Q^n or (Q^-1)^n = (Q^n)^-1. These values have the same x-coordinate. Thus, the x-coordinate of a point P^i can be computed from the x-coordinate of a point P by computing one of the possible values of the y coordinate of P, then computing the ith power of P, and then ignoring the y-coordinate of that result.

In general, it is possible to compute a square root modulo p by using Shanks' method [K1981v2]; simple methods exist for some values of p. When p = 3 (mod 4), the square roots of z mod p are w and -w mod p, where

```
w = z ^ ((p+1)/4) (mod p);
```

this observation is due to Lehmer [L1969]. When p satisfies this property, y can be computed from the curve equation, and either y = w or y = -w mod p, where

$$w = (x^3 + a*x + b)^{((p+1)/4)} \pmod{p}.$$

Square roots modulo p only exist for a quadratic residue modulo p, [R1988]; if z is not a quadratic residue, then there is no number w such that $w^2 = z \pmod{p}$. A simple way to verify that z is a quadratic residue after computing w is to verify that $w * w = z \pmod{p}$. If this relation does not hold for the above equation, then the value x is not a valid x-coordinate for a valid elliptic curve point. This is an important consideration when ECDH is used with compact output; see Section 10.3.

The primes used in the P-256, P-384, and P-521 curves described in [RFC5903] all have the property that $p = 3 \pmod{4}$.

Appendix D. Example ECC Parameter Set

For concreteness, we recall an elliptic curve defined by Solinas and Fu in [RFC5903] and referred to as P-256, which is believed to provide a 128-bit security level. We use the notation of Section 3.3, and express the generator in the affine coordinate representation g=(gx,gy), where the values gx and gy are in Fp.

p: FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF

a: - 3

b: 5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B

n: FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

gx: 6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296

gy: 4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5

Note that p can also be expressed as

$$p = 2^{256}-2^{224}+2^{192}+2^{96}-1.$$

Appendix E. Additive and Multiplicative Notation

The early publications on elliptic curve cryptography used multiplicative notation, but most modern publications use additive notation. This section includes a table mapping between those two conventions. In this section, a and b are elements of an elliptic curve group, and N is an integer.

Multiplicative Notation	Additive Notation
multiplication	addition
a * b	a + b
squaring	doubling
a * a = a^2	a + a = 2a
exponentiation	scalar multiplication
a^N = a * a * ... * a	Na = a + a + ... + a
inverse	inverse
a^-1	-a

Appendix F. Algorithms

This section contains a pseudocode description of the elliptic curve group operation. Text that follows the symbol "//" is to be interpreted as comments rather than instructions.

F.1. Affine Coordinates

To an arbitrary pair of elliptic curve points P and Q specified by their affine coordinates P=(x1,y1) and Q=(x2,y2), the group operation assigns a third point R = P*Q with the coordinates (x3,y3). These coordinates are computed as follows:

```
if P is (@,@),
   R = Q
else if Q is (@,@),
   R = P
else if P is not equal to Q and x1 is equal to x2,
   R = (@,@)
else if P is not equal to Q and x1 is not equal to x2,
   x3 = ((y2-y1)/(x2-x1))^2 - x1 - x2 mod p and
   y3 = (x1-x3)*(y2-y1)/(x2-x1) - y1 mod p
else if P is equal to Q and y1 is equal to 0,
   R = (@,@)
else    // P is equal to Q and y1 is not equal to 0
   x3 = ((3*x1^2 + a)/(2*y1))^2 - 2*x1 mod p and
   y3 = (x1-x3)*(3*x1^2 + a)/(2*y1) - y mod p.
```

From the first and second case, it follows that the point at infinity is the neutral element of this operation, which is its own inverse.

From the curve equation, it follows that for a given curve point P = (x,y) distinct from the point at infinity, (x,-y) also is a curve point, and from the third and the fifth case it follows that this is the inverse of P, P^-1.

Note: The fifth and sixth case are known as "point squaring".

F.2. Homogeneous Coordinates

An elliptic curve point (x,y) (other than the point at infinity (@,@)) is equivalent to a point (X,Y,Z) in homogeneous coordinates (with X, Y, and Z in Fp and not all three being zero at once) whenever x=X/Z and y=Y/Z. "Homogenous coordinates" means that two triples (X,Y,Z) and (X',Y',Z') are regarded as "equal" (i.e., representing the same point) if there is some nonzero s in Fp such that X'=s*X, Y'=s*Y, and Z'=s*Z. The point at infinity (@,@) is regarded as equivalent to the homogenous coordinates (0,1,0), i.e., it can be represented by any triple (0,Y,0) with nonzero Y in Fp.

Let P1=(X1,Y1,Z1) and P2=(X2,Y2,Z2) be points on the elliptic curve, and let u = Y2 * Z1 - Y1 * Z2 and v = X2 * Z1 - X1 * Z2.

We observe that the points P1 and P2 are equal if and only if u and v are both equal to zero. Otherwise, if either P1 or P2 are equal to the point at infinity, v is zero and u is nonzero (but the converse implication does not hold).

Then, the product P3=(X3,Y3,Z3) = P1 * P2 is given by:

```
  if P1 is the point at infinity,
     P3 = P2
  else if P2 is the point at infinity,
     P3 = P1
  else if u is not equal to 0 but v is equal to 0,
     P3 = (0,1,0)
  else if both u and v are not equal to 0,
     X3 = v * (Z2 * (Z1 * u^2 - 2 * X1 * v^2) - v^3)
     Y3 = Z2 * (3 * X1 * u * v^2 - Y1 * v^3 - Z1 * u^3) + u * v^3
     Z3 = v^3 * Z1 * Z2
  else    // P2 equals P1, P3 = P1 * P1
      w = 3 * X1^2 + a * Z1^2
     X3 = 2 * Y1 * Z1 * (w^2 - 8 * X1 * Y1^2 * Z1)
     Y3 = 4 * Y1^2 * Z1 * (3 * w * X1 - 2 * Y1^2 * Z1) - w^3
     Z3 = 8 * (Y1 * Z1)^3
```

It thus turns out that the point at infinity is the identity element and for P1=(X,Y,Z) not equal to this point at infinity, P2=(X,-Y,Z) represents P1^-1.

Authors' Addresses

David A. McGrew
Cisco Systems
510 McCarthy Blvd.
Milpitas, CA 95035
USA

Phone: (408) 525 8651
EMail: mcgrew@cisco.com
URI: http://www.mindspring.com/~dmcgrew/dam.htm

Kevin M. Igoe
National Security Agency
Commercial Solutions Center
United States of America

EMail: kmigoe@nsa.gov

Margaret Salter
National Security Agency
9800 Savage Rd.
Fort Meade, MD 20755-6709
USA

EMail: msalter@restarea.ncsc.mil