Network Workign Group
Internet-Draft
Intended status: Informational
Expires: June 9, 2014

A. Jivsov
Symantec Corporation

December 6, 2013

# Compact representation of an elliptic curve point
# draft-jivsov-ecc-compact-03

Abstract

This document defines a format for efficient storage representation of an elliptic curve point over prime fields, suitable for use with any IETF format or protocol.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on June 9, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction . . . . . 3
2. Conventions used in this document . . . . . 3
3. Overview of the compact representation in IETF protocols . . . 3
4. The definition of the compact representation . . . . . 4
4.1. Encoding and decoding of an elliptic curve point . . . . . 5
4.2. The algorithms to generate a key pair . . . . . 5
4.2.1. The black box key generation algorithm . . . . . 6
4.2.2. The deterministic key generation algorithm . . . . . 6
4.3. The efficient square root algorithm for p=4*k+3 . . . . . 7
5. Interoperability considerations . . . . . 7
6. IANA Considerations . . . . . 8
7. Security Considerations . . . . . 8
8. References . . . . . 10
8.1. Normative References . . . . . 10
8.2. Informative References . . . . . 10
Appendix A. Sample code change to add compliant key generation to libgcrypt and openssl . . . . . 11
A.1. libgcrypt changes . . . . . 11
A.2. OpenSSL changes . . . . . 12
Author's Address . . . . . 13

## 1. Introduction

The National Security Agency (NSA) of the United States specifies elliptic curve cryptography (ECC) for use in its [SuiteB] set of algorithms. The NIST elliptic curves over the prime fields [FIPS-186], which include [SuiteB] curves, or the Brainpool curves [RFC5639] are the examples of curves over prime fields.

This document provides an efficient format for compact representation of a point on an elliptic curve over a prime field. It is intended as an open format that other IETF protocols can rely on to minimize space required to store an ECC point. This document complements the [RFC6090] with the on-the-wire definition of an ECC point.

One of the benefits of the ECC is the small size of field elements. The compact representation reduces the encoded size of an ECC element in half, which can be a substantial saving in cases such as encryption of a short message sent to multiple recipients.

## 2. 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 [RFC2119].

## 3. Overview of the compact representation in IETF protocols

IETF protocols often use the [SEC1] representation of a point on an elliptic curve, which is a sequence of the following fields:

| Field | Description |
|---|---|
| B0 | {02, 03, 04}, where 02 or 03 represent a compressed point (x only), while 04 represents a complete point (x,y) |
| X | x coordinate of a point |
| Y | y coordinate of a point, optional (present only for B0=04) |

SEC1 point representation

The [SEC1] is an example of a general-purpose elliptic curve point compression. The idea behind these methods is the following:

- For the given point P=(x,y) the y coordinate can be derived from x by solving the corresponding elliptic curve equation.

- There are two possible y coordinates for any x of a given P
- The either of the two possibilities for y is encoded in some way in the compressed representation

There are a few undesirable properties of the above representation:

- The requirement to store one bit to identify the 'y' means that the whole byte is required.
- For most well-known elliptic curves the extra byte removes the power of two alignment for the encoded point.
- The requirement for the balanced security calls for the ECC curve size to be equal the hash output size, yet the storage length of the ECC point is equal to the hash output size + 1.
- The encoded point is not a multi-precision integer, but a structured sequence of bytes. For example, special wording is required to define the encoding of the [FIPS-186] P-521 to clarify how odd number of bits for x and y, or a bit representing y, are packed into bytes.
- Some protocols, such as ECDH, don't depend on the exact value of the y. It is unnecessary to track the precise point P=(x,y) in such protocols.

## 4. The definition of the compact representation

This document is an improvement to the idea by [Miller] to not transmit the y coordinate of an ECC point in the elliptic curve Diffie-Hellman (ECDH) protocol.

We will use the following notations for the ECC point Q and the features of the corresponding elliptic curve:

Q = k*G, where

Q = (x,y) is the point on an elliptic curve (the canonical represenation)

k - the private key (a scalar)

G - the elliptic curve generator point

y^2 = x^3 + a*x + b is the standard Weierstrass equation linking x and y

p - the order of the underlying finite field to which x and y belong

Ord - the order of the elliptic curve field, i.e. the number of points on the curve ( Ord*G = O, where O is the identity element )

Q is a point that we need to represent in the compact form. The integer operations considered in this document are performed modulo prime p and "(mod p)" is assumed in every formula with x and y.

The steps to create and interpret the compact representation of a point are described next. A special key generation algorithm is needed to make them possible, defined later in Section 4.2.

## 4.1. Encoding and decoding of an elliptic curve point

Encoding: Given the canonical representation of Q=(x,y), return the x as its compact representation.

Decoding: Given the compact representation of Q, return canonical representation of Q=(x,y) as follows:

1. y' = sqrt( x^3 + a*x + b ), where y'>0

2. y = min(y',p-y')

3. Q=(x,y) is the canonical representation of the point

Recall that the x is an element in the underlying finite field, represented by an integer. Its precise encoding SHOULD be consistent with encoding of other multi-precision integers in the application, for example, it would be the same encoding as used for the r or s integer that is a part of the DSA signature and it is typically a sequence of big-endian bytes.

The efficient algorithm to recover y for [SuiteB] or the Brainpool curves [RFC5639], among others, is given in Section 4.3.

min(y,p-y) can be calculated with the help of the pre-calculated value p2=(p-1)/2. min(y,p-y) is y if y<=p2 and p-y otherwise.

The efficient encoding and decoding algorithms are possible with the special key generation algorithm, defined next.

## 4.2. The algorithms to generate a key pair

This document specifies two algorithms, called the "black box" and the "deterministic" key generation algorithms, to generate a key pair

{k, Q=k*G=(x,y)}, where k is the private key and Q=(x,y) is the public key. A key pair generated according to the requirements in this section is called a compliant key pair, and the public key of such a key pair -- a compliant public key. A compliant public key Q=(x,y) allows compact representation as x, as defined in Section 4.1.

Both key generation algorithms can be built with any general purpose key generation algorithm which would be needed in any ECC implementation that generates keys, regardless of the support for any method defined in this document. Such a general purpose key generation algorithm is referred in this section as "KG".

The black box algorithm works in scenarios when the KG doesn't allow any adjustments to the private key. The disadvantage of this algorithm is that multiple KGs may be needed to generate a single key pair {k, Q}. The deterministic algorithm is similar, except that it is allowed to perform a simple and fast modification to the private key after the KG. The advantage of the second algorithm is performance, in particular, the guarantee that only a single KG is needed.

### 4.2.1. The black box key generation algorithm

The following algorithm calculates a key pair {k, Q=k*G=(x,y)}, where k is the private key and Q=(x,y) is the public key.

Black box generation:

1. Generate a key pair {k, Q=k*G=(x,y)} with KG

2. if( y != min(y,p-y) ) goto step 1

3. output {k, Q=(x,y)} as a key pair

Note that the step 1 is a general purpose key generation algorithm, such as an algorithm compliant with [NIST-SP800-133]. Step 1 assumes neither changes to existing key generation methods nor access to the private key in clear.

The expected number of iterations in the loop in the above algorithm is 2. The step 2 is not needed for the ECDH keys.

### 4.2.2. The deterministic key generation algorithm

The following algorithm calculates a key pair {k, Q=k*G=(x,y)}, where k is the private key and Q=(x,y) is the public key.

Deterministic generation:

1. Generate a key pair {k, Q=k*G=(x,y)} with KG

2. if( y != min(y,p-y) ) k = Ord - k; y = p - y

3. output {k, Q=(x,y)} as a key pair

The step 2 is not needed for the ECDH keys.

### 4.3. The efficient square root algorithm for p=4*k+3

When p mod 4 = 3, as is the case of [SuiteB] and the Brainpool curves [RFC5639], there is an efficient square root algorithm to recover the y, as follows:

Given the compact representation of Q as x,

y2 = x^3 + a*x + b

y' = y2^((p+1)/4)

y = min(y',p-y')

Q=(x,y) is the canonical representation of the point

See [Lehmer] for details.

## 5. Interoperability considerations

The compact representation described in this document allows two-phase introduction.

First, key pairs must be generated as defined in Section 4.2 to allow compact representation. No accompanied changes are needed elsewhere to use these keys. This allows safe deployment of the new key generation, which, in turn, allows encoding and decoding of compact representation, possibly at a later time.

Finally, the encoding of public keys in the new compact representation format can be enabled after there is confidence in the universal support of new compact representation. This event would not need to change any private key material, only public key representation.

The above two phases can be implemented at once for new formats.

Most ECC cryptographic protocols, such as ECDSA [FIPS-186], are intended to work with persistently stored public keys that are generated as fresh key pairs, as opposed to some derivation function that transforms an ECC point. The algorithm described in Section 4.2 is possible in all these cases. Furthermore, a typical instantiation of the ECDH protocol, such as ECDH specified in [NIST-SP800-56A], makes any ECC key used in a DH key key agreement automatically compliant for the purpose of this specification ( as noted in the Section 4.2 ). The algorithm in Section 4.2 will even work for secure devices that never reveal the private key, such as smartcards or Hardware Security Modules. A public key that is generated according to the Section 4.2 can be used without limitations in existing protocols that use ECC points encoded in other ways, such as [SEC1], with compression or not, with the added advantage that the keys generated according to the method in Section 4.2 will allow the Section 4.1 encoding.

## 6. IANA Considerations

This document defines a low-level format that may be suitable for a wide range of applications. However, it is responsibility of the application that adopts this format to define the IDs that will enable the ECC compact point representation in that application.

A new ID may not be always necessary. For example, an application that currently allows the [SEC1] encoding may allow the compact representation defined in this document as an extension to the [SEC1] as follows. Consider the encoding of a compressed [FIPS-186] P-256 point, for example. The [SEC1] compressed representation of a P-256 point will always occupy exactly 33 bytes. On the other hand, the compact representation defined in this document will never exceed 32 bytes (it may occupy fewer that 32 bytes when the most significant byte has happened to be zero). This size will allow reliable discrimination between two encoding formats.

## 7. Security Considerations

The key pair generation process in Section 4.2 excludes exactly half of the points on the elliptic curve. What is left is the subset of points suitable for compact representation. The filtering of points is based on a public criteria that are applied to the public output of the ECC one-way function.

The set of Ord points on the elliptic curve can be subdivided as follows. First, remove the point O, which leaves Ord-1 points. Of these points there are exactly (Ord-1)/2 points that have unique x

coordinate. This document specifies a method to form the (Ord-1)/2 of points, each having a unique x coordinate. These points are called compliant public keys in Section 4.2.

For any two public keys P=(x,y) and P=(x,y') there is up to one bit of entropy in y' v.s. y and this information is public. This bit of entropy doesn't contribute to the difficulty of the underlying hard problem of the ECC: the elliptic curve discrete logarithm problem (ECDLP).

It will be shown next that breaking the ECDLP with a key generated according to Section 4.2 is not easier than breaking the ECDLP with a key obtained through a standard key generation algorithm, referred to as the KG algorithm in the Section 4.2.

Let us assume that there is an algorithm A that solves the ECDLP for the KG. The algorithm A can be transformed into the algorithm A' as follows.

- If P=(x,y) is a compliant public key, the ECDLP is solved with A for the point (x,y): the result is k, such that k*G=(x,y)

- If P=(x,y) is not a compliant public key, the ECDLP is solved with A for the point (x,p-y); assuming the output produced by A is k, the output produced by A' is set to (Ord-k). Note that (Ord-k)*G = (x,y), which means that the output of A' is the correct solution.

A' is equivalent to A. The complexity of one additional substraction in the prime field is negligible even to the complexity of a single elliptic curve addition. Observe that A' works for all public keys by performing the actual work only on compliant public keys.

If we now consider only the compliant public keys, which cuts the number of points in half, we observe that the ECDLP solving algorithm A' doesn't get to break fewer public keys. This concludes the proof.

The same result can be observed based on the details of the current state of the art attacks on the ECDLP. These attacks use Pollard's rho algorithm, which uses the collision search in the sequence(s) of generated points with the goal to produce the points P1=(x1,y1) and P2=(x2,y2), such that x1=x2 and y1=y2. The match in the x coordinate is the sufficient event for the successful attack. After this event has ocurred, the sequence(s) that led to x1=x2 collision can be adjusted in a constant number of steps to ensure that y1=y2, if this is not already the case. Furthermore, collision search requires the storage of candidates for the collision. It's wasteful to store (x,y) v.s. storing x and only calculating y when the collision in x

is detected. Thus, the ECDLP attack does not benefit from the unpredictability of the y.

Finally, note that a common design feature of an ECDH-based system is not to depend on the y coordinate, such as the one defined in the [NIST-SP800-56A]. Thus, the security of the system is unaffected if we fix either of the two possibilities for the point with the given x coordinate.

## 8. References

### 8.1. Normative References

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

### 8.2. Informative References

[FIPS-186] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS 186-3, June 2009, <http://csrc.nist.gov/publications/PubsSPs.html>.

[Lehmer] Lehmer, D., "Computer technology applied to the theory of numbers", 1969.

[Miller] Miller, V., "Use of elliptic curves in cryptography", Proceedings Lecture notes in computer sciences; 218 on Advances in cryptology -- CRYPTO 85, June 1986.

[NIST-SP800-133] National Institute of Standards and Technology, "Recommendation for Cryptographic Key Generation", SP 800-133, November 2012, <http://csrc.nist.gov/publications/PubsSPs.html>.

[NIST-SP800-56A] National Institute of Standards and Technology, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", SP 800-56A Revision 1, March 2007, <http://csrc.nist.gov/publications/PubsSPs.html>.

[RFC5639] Lochter, M. and J. Merkle, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation", RFC 5639, March 2010.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[SEC1] STANDARDS FOR EFFICIENT CRYPTOGRAPHY, "SEC 1: Elliptic Curve Cryptography", September 2000, <www.secg.org/collateral/sec1_final.pdf>.

[SuiteB] National Security Agency, "NSA Suite B Cryptography", March 2010, <http://www.nsa.gov/ia/programs/suiteb_cryptography/>.

[openssl] Jivsov, A., "An enhancement to EC key generation to enable compact point representation", June 2013, <http://rt.openssl.org/Ticket/History.html?id=3069>.

## Appendix A. Sample code change to add compliant key generation to libgcrypt and openssl

The complete changes that were needed to make libgcrypt library generate a compliant key are shown inline in this section, followed by pending changes to OpenSSL.

### A.1. libgcrypt changes

In the following changes, the Q is the initial public key, G generator, and d is the corresponding private key. "-" prefix marks the two lines that were replaced with the lines starting with "+". Lines starting with "+" represent the code that adds compliant key generation to libgcrypt.

```
@@ generate_key (ECC_secret_key *sk,
     unsigned int nbits,
     const char *name,
   point_set (&sk->E.G, &E.G);
   sk->E.n = mpi_copy (E.n);
   point_init (&sk->Q);
-  point_set (&sk->Q, &Q);
-  sk->d    = mpi_copy (d);
+
+  /* We want the Q=(x,y) be a "compliant key" in terms of the
+   * http://tools.ietf.org/html/draft-jivsov-ecc-compact,
+   * which simply means that we choose either Q=(x,y) or -Q=(x,p-y)
+   * such that we end up with the min(y,p-y) as the y coordinate.
+   * Such a public key allows the most efficient compression: y can
+   * simply be dropped because we know that it's a minimum of
+   * the two possibilities without any loss of security.
+   */
```

```
+  {
+    gcry_mpi_t x, p_y, y;
+    const unsigned int nbits = mpi_get_nbits (E.p);
+
+    x = mpi_new (nbits);
+    p_y = mpi_new (nbits);
+    y = mpi_new (nbits);
+
+    if (_gcry_mpi_ec_get_affine (x, y, &Q, ctx))
+      log_fatal ("ecgen: Failed to get affine coordinates for Q\n");
+
+    mpi_sub( p_y, E.p, y );  /* p_y = p-y */
+
+    if( mpi_cmp( p_y /*p-y*/, y ) < 0 )  {  /* is p-y < p ? */
+      gcry_mpi_t z = mpi_copy( mpi_const (MPI_C_ONE) );
+      /* we need to end up with -Q; this assures that new Q's y
+       * is the smallest one */
+      sk->d = mpi_new (nbits);
+      mpi_sub( sk->d, E.n, d );  /* d = order-d */
+      /* log_mpidump ("ecgen d after ", sk->d); */
+      gcry_mpi_point_set (&sk->Q, x, p_y/*p-y*/, z);  /* Q = -Q */
+      if (DBG_CIPHER)
+      {
+        log_debug   ("ecgen converted Q to a compliant point\n");
+      }
+      mpi_free (z);
+    }
+    else
+    {
+      /* no change is needed exactly 50% of the time: just copy */
+      sk->d = mpi_copy (d);
+      point_set (&sk->Q, &Q);
+      if (DBG_CIPHER)
+      {
+        log_debug   ("ecgen didn't need to convert Q to "
+                     "a compliant point\n");
+      }
+    }
+    mpi_free (x);
+    mpi_free (p_y);
+    mpi_free (y);
+  }
```

## A.2. OpenSSL changes

OpenSSL changes are even more compact than in Appendix A.1. They are tracked at [openssl].

## Author's Address

Andrey Jivsov
Symantec Corporation

Email: openpgp@brainhub.org