Independent Stream
Internet-Draft
Intended status: Informational
Expires: 14 December 2025

M. Schanzenbach
Fraunhofer AISEC
P. Fardzadeh
Technische Universit辰t M端nchen

12 June 2025

The HPKE Elligator KEM
draft-schanzen-hpke-elligator-kem-01

Abstract

This document contains the GNUnet communicator specification.

This document defines the normative wire format of communicator protocols, cryptographic routines and security considerations for use by implementers.

This specification was developed outside the IETF and does not have IETF consensus. It is published here to inform readers about the function of GNUnet communicators, guide future communicator implementations, and ensure interoperability among implementations including with the pre-existing GNUnet implementation.

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 https://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 14 December 2025.

Copyright Notice

Table of Contents

1. Introduction

Diffie-Hellman-based KEMs (DHKEMs) allow us to securely establish a secret between two parties. However, an observer can quickly identify the exchanged encapsulation in DHKEMs as public keys. In the presence of a passive eavesdropping attacker, packets could drop based on this information, preventing communication between peers as outlined in [BHKL13]. The presented solution in [BHKL13] is called "Elligator" and allows us to produce random-looking representations of curve points. This leaves an attacker with fewer options: either do nothing or intercept most random-looking packets, thereby potentially disrupting a large part of today's internet communication.

In the case of Montgomery curves, such as Curve25519, a point [X, Y] on that curve (e.g., the ephemeral public key) follows the equation Y^2 = X^3 + A * X^2 + X mod P, where A and P are parameters for Curve25519 specified in Section 4.1 of [RFC7748]. For any valid x-coordinate, the left side of the equation is always a quadratic number. An attacker could read the x-coordinate and verify if this property holds. While this property holds for any valid Curve25519 point, it only holds for a random number in about 50% of the cases.

By observing multiple communication attempts, an attacker can be sure that curve points are being sent if the property consistently holds. To circumvent this attack, curve points should be encoded into property-less numbers, making valid and invalid curve points indistinguishable to an outside observer. The Elligator encoding function (also known as the "inverse map") and decoding function (also known as the "direct map") implement this feature.

In this document, we define and use an Elligator transformation for X25519 curve points based on the Curve25519 transformations in [BHKL13] to be used in a Key Encapsulation Mechanim (KEM) for use in HPKE [RFC9180] and its security considerations for use by implementers.

This specification was developed outside the IETF and does not have IETF consensus. It is published here to guide implementers and ensure interoperability among implementations.

1.1. Requirements Notation

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Notation

We use the notation and terminology of [RFC9180] throughout this document. In addition, we define:

coinFlip() A helper function that returns "heads" or "tails". Each result is returned with a likelihood of 50%.

3. Elligator DHKEM

The Elligator HPKE DHKEM utilizes Elligator to encode and decode the ephemeral public keys as described in Section 5 of [BHKL13]. We define our KEM analogous to [RFC9180] Section 4. The kem_id in the suite_id for the Elligator KEM is TBD

The ExtractAndExpand(), Encap() and Decap() functions (and their authenticated variants) can remain unchanged and MUST be implemented as defined in Section 4.1 of [RFC9180]. The serialization functions SerializePublicKey and DeserializePublicKey are defined in the following for Curve25519.

3.1. GenerateKeyPair()

First, not all X25519 key pairs are suitable candidates for Elligator. In particular, not all Curve25519 points have the property that the Elligator encoding and subsequent decoding result in the original point (See Section 4.1 for details). To create a Curve25519 point that can be used with Elligator, one needs to find a curve point for which this property holds. When generating a key pair suitable for Elligator, the general idea is to create both a random high-order point and a low-order point. Adding them together results in a curve point that is evenly distributed on the whole curve. One heuristic is to generate random key pairs until one such point is found:

Let G be the generator of the prime order group of Ed25519 and H the generator of the low order subgroup of Ed25519. EdToCurve() is a function that converts Ed25519 points to their corresponding Curve25519 points. We define "GenerateElligatorKeyPair()" as follows:

```
GenerateElligatorKeyPair():
  VALID := 0
  while(!VALID):
    skX := random(Nsk)
    skX[0] &= 24
    skX[31] &= 127
    skX[31] |= 64
    E_high := skX * G
    E_low := (skX mod 8) * H
    E := E_high + E_low
    pkX := EdToCurve(E)
    if ElligatorDec(ElligatorEnc(pkX)) == pkX:
      return (skX,pkX)
```

We operate on the Edwards representation for the key generation because the necessary low-order point additions are more efficient in most implementations than they would be in their Montgomery form. A conversion from Edwards to their birationally equivalent Montgomery form is always possible and found in most cryptographic library implementations.

The GenerateKeyPair algorithm MUST be implemented to produce a key pair suitable for Elligator. The GenerateElligatorKeyPair() algorithm defined here is RECOMMENDED. Other, more efficient key generation algorithms MAY be used.

3.2. SerializePublicKey()

The serialization functions incorporate the Elligator inverse and direct map functions to obfuscate a curve point. The Elligator literature calls the obfuscated curve point a "representative". In the following, the "representative" is named pkXm, where X stands for any of the roles defined in [RFC9180].

Let A and P be the parameters for Curve25519 as specified in section 4.1 of [RFC7748]. Further, let X be any valid x-coordinate of a Curve25519 point, L() a function that computes the Legendre symbol of a field element with regard to the odd prime P. Let sqrt() denote the square root operation within the field. As each square number has two roots, we need to define the notion of positive and negative numbers. There are multiple valid ways to partition the field elements. For this Elligator, we follow the recommendations of the paper in section 5.1 of [BHKL13] and choose {0,..., (P-1)/2} as the set of positive numbers, and consequently {(P-1)/2 + 1,...,P-1} as the set of the negative numbers. Both Elligator's inverse and direct map require us to define a constant non-square number of the finite field. Let U := sqrt(-1) be this number. The resulting serialization algorithm for the HPKE KEM can then be described as:

```
SerializeElligatorPublicKey(pkX):
  if coinFlip() == 1:
    pkXm :=  sqrt(-pkX / ((pkX + A) * U))
  else:
    pkXm :=  sqrt(-(pkX + A) / (U * X))
  if coinFlip() == 1:
    pkXm[31] |= 128
  if coinFlip() == 1:
    pkXm[31] |= 64
  return pkXm
```

Note that SerializeElligatorPublicKey(pkX) represents Elligator's inverse map with a slight modification: The resulting representative of the inverse map is strictly smaller than 2^254 - 9. Therefore, the most and second most significant bits are always zero, an obvious property an attacker could observe. We avoid this problem by randomly flipping both bits. The target peer will ignore these bits after reception by setting those bits back to zero. Similarly, as there are always two roots for each square number, we randomly select one or the other upon each serialization.

3.3. DeserializePublicKey()

```
DeserializeElligatorPublicKey(pkXm):
  pkXm[31] &= 63
  V := -A / (1 + U * pkXm^2)
  E := L(V^3 + A * V^2 + V)
  pkX := E * V - (1 - E)(A / 2)
  return pkX
```

Note that DeserializeElligatorPublicKey(pkX) represents Elligator's direct map. We must, and can safely, clear the most and second most significant bits that were set randomly as elaborated above before reconstructing the square root.

4. Security and Privacy Considerations

4.1. Elligator

Most modern implementations of Curve25519 only generate points from its prime subgroup to circumvent known attacks for which points not within the prime subgroup are susceptible. Those attacks are not an issue in our case as we use the ephemeral secret key only once for computing key material. The exclusive use of the prime subgroup is a recognizable property that an outside observer can easily detect, even when using the encoding function. An attacker could decode the suspected parts of packets to the corresponding Curve25519 points and check if the resulting points are always in the prime subgroup. To circumvent this attack, we must randomly choose the ephemeral key pair from the whole curve as defined in "GenerateElligatorKeyPair()".

The intuition behind elligator is based on the following idea: The direct map (here DeserializeElligatorPublicKey(r)) expects a random field element r. The value r is mapped to two values, for which only one coordinate is a valid Curve25519 x-coordinate. One can show the pair of values V_1 := -A / (1 + U * r^2) and V_2 := -A * U * r^2 (1 + U * r^2) always fulfill this property, based on the fact that U is a non-square number in the field. The direct map first computes V_1 and checks if V_1 fulfills the curve equation. If it does, it returns V_1; otherwise, V_2 is computed and returned instead. The inverse map, (here SerializeElligatorPublicKey(V)) reverses this process and computes the corresponding field element of a valid x-coordinate. Note that for encode and decode public keys, we reverse the call order of the direct and inverse map: First, the inverse map is called on the ephemeral public key to retrieve the representative r. This representative is then send to the receiving peer, who calls the direct map to retrieve the corresponding public key.

Note that both for a value r and its negative counterpart -r (in the finite field), the inverse map function will result in the same x-coordinate. Moreover, for two different valid x-coordinates, the resulting representatives of the corresponding encoding calls are different. Conversely, we can't decode both representatives back to their original x-coordinate. This is why the sender eventually tries several random key pairs in GenerateElligatorKeyPair() to create a valid public key that can be used for a key exchange. Also note that this effectively reduces the entropy of our public keys by 1 bit, which is tolerable.

In [BHKL13], Elligator's inverse map takes the sign of y-coordinate as an additional input parameter. Its value determines which of the two terms is used instead of our random selection. Due to known attacks, we also skip the calculation of the corresponding y-coordinate in the decoding function. We omitted the y-coordinate part of both functions because Curve25519 points are solely represented by their x-coordinate in modern cryptosystems. Nevertheless, the desired feature of Elligator is still ensured.

4.2. Combination with AEAD Encryption

When using the Elligator KEM in combination with AEAD encryption schemes, care must be taken to ensure that the ciphertext produced by the AEAD cipher is also indistinguishable from random. The AEAD schemes listed in [RFC9180] use GCM and Poly1305 authentication tags, which should result in ciphertexts that are indistinguishable from random. However, future AEAD schemes and, in particular, their authenticators may not exhibit the same cryptographic properties. This should be considered when assembling HPKE suites with the Elligator KEM.

5. IANA Considerations

This document defines a new KEM as allowed in [RFC9180]. The "HPKE KEM Identifiers" registry is requested to be updated with the values from Table 1. This section may be removed on publication as an RFC.

Value	KEM	Nsecret	Nenc	Npk	Nsk	Auth	Reference
TBD	DHKEM(X25519+Elligator, HKDF-SHA256)	32	32	32	32	yes	This.I-D

Table 1: KEM IDs

6. Implementation and Deployment Status

There is one implementation conforming to this specification, written in C. The implementation is part of [GNUnet] and represents the original and reference implementation.

The basic Elligator primitives GenerateKeyPair(), SerializePublicKey() and DeserializePublicKey() are present in [GNUnetElligator]. The corresponding KEM primitives are part of [GNUnetHPKE].

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/info/rfc9180>.

8. Informative References

[BHKL13] Bernstein, D.J., Hamburg, M., Krasnova, A., and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings", August 2013, <https://eprint.iacr.org/2013/325.pdf>.

[GNUnet] GNUnet e.V., "gnunet.git - GNUnet core repository", 2023, <https://git.gnunet.org/gnunet.git>.

[GNUnetElligator] Schanzenbach, M. and P. Fardzadeh, "gnunet.git - Elligator primitives implementation in GNUnet core repository", 2023, <https://git.gnunet.org/gnunet.git/tree/src/lib/util/crypto_elligator.c>.

[GNUnetHPKE]
Schanzenbach, M. and P. Fardzadeh, "gnunet.git - HPKE Primitive implementation in GNUnet core repository", 2023, <https://git.gnunet.org/gnunet.git/tree/src/lib/util/crypto_hpke.c>.

Appendix A. Elligator implementation

This section provides a test vector for the Elligator KEM and should aid in verifying implementations. Note that Elligator has two parameters: the set of positive and negative numbers, and a non-square number U within the finite field, as described in section 5.1 of [BHKL13]. The displayed test vectors assume that the set of positive numbers is defined as {0,...,(P-1)/2}, the set of negative numbers as {(P-1)/2 + 1,...,P−1} and U is the non-square number sqrt(-1). The depicted coin flips are used in the order of the coinFlip() calls in SerializeElligatorPublicKey(pkX), namely coin flip 1 for choosing the pkXm term, coin flip 2 for the MSB and coin flip 3 for the second MSB. Unless indicated otherwise, the test vectors are provided as little-endian hexadecimal byte arrays.

A.1. Elligator KEM

coin flip 1: 0
coin flip 2: 1
coin flip 3: 1
pkEm:
3f73ee0dd1970ff957f7ec15e0b5151166be3046e6a8b0ee53beca395b74e42c

skEm:
09395966d6d1c493b9917dd12c8dd24e2c05c081c98a67eb2d6dff622ec9c069

skRm:
f33887a8562dad5151e9289a0afa1301ccc698917850d56ea409a9949497baa4

pkRm:
3febadac122d397725ff580f6ce9a3e1c1c4a7de19807f13d383f2f9b6467136

enc:
da0f7edaefed18a99f0b73a789e51c4c6e80664190ae3c8ae4e95b9d926a34f7

key:
46eff65b5313f41fbaffc7adf98f5df03ab4e4f46ae62a2c7ecbe1f0ae83280b

Authors' Addresses

Martin Schanzenbach
Fraunhofer AISEC
Lichtenbergstrasse 11
85748 Garching
Germany
Email: martin.schanzenbach@aisec.fraunhofer.de

Pedram Fardzadeh
Technische Universitt Mnchen
Boltzmannstrasse 3
85748 Garching
Germany
Email: pedram.fardzadeh@tum.de