Workgroup: Network Working Group
Internet-Draft:
draft-irtf-cfrg-hash-to-curve-06
Published: 9 March 2020
Intended Status: Informational
Expires: 10 September 2020
Authors: A. Faz-Hernandez S. Scott N. Sullivan
Cloudflare Cornell Tech Cloudflare
R.S. Wahby C.A. Wood
Stanford University Apple Inc.

Hashing to Elliptic Curves

Abstract

This document specifies a number of algorithms that may be used to encode or hash an arbitrary string to a point on an elliptic curve.

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 10 September 2020.

Copyright Notice

Table of Contents

1. Introduction

Many cryptographic protocols require a procedure that encodes an arbitrary input, e.g., a password, to a point on an elliptic curve. This procedure is known as hashing to an elliptic curve. Prominent examples of cryptosystems that hash to elliptic curves include Simple Password Exponential Key Exchange [J96], Password Authenticated Key Exchange [BMP00], Identity-Based Encryption [BF01] and Boneh-Lynn-Shacham signatures [BLS01].

Unfortunately for implementors, the precise hash function that is suitable for a given scheme is not necessarily included in the description of the protocol. Compounding this problem is the need to pick a suitable curve for the specific protocol.

This document aims to bridge this gap by providing a thorough set of recommended algorithms for a range of curve types. Each algorithm conforms to a common interface: it takes as input an arbitrary-length byte string and produces as output a point on an elliptic curve. We provide implementation details for each algorithm, describe the security rationale behind each recommendation, and give guidance for elliptic curves that are not explicitly covered.

This document does not cover rejection sampling methods, sometimes known as "try-and-increment" or "hunt-and-peck," because the goal is to describe algorithms that can plausibly be made constant time. Use of these rejection methods is NOT RECOMMENDED, because they have been a perennial cause of side-channel vulnerabilities.

1.1. How to use this document

This document is intended for use by both implementors and protocol designers.

For implementors, the necessary and sufficient level of specification is a hash-to-curve suite, which fixes all of the parameters listed in Section 8, plus a domain separation tag (Section 3.1). Starting from working operations on the target

elliptic curve and its base field, a hash-to-curve suite requires implementing the specified encoding function (Section 3), its constituent subroutines (Section 5, Section 6, Section 7), and a few utility functions (Section 4).

Correspondingly, designers specifying a protocol that requires hashing to an elliptic curve should either choose an existing hash-to-curve suite or specify a new one (see Section 8.8). In addition, designers should choose a domain separation tag following the guidelines in Section 3.1.

1.2. Requirements

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].

2. Background

2.1. Elliptic curves

The following is a brief definition of elliptic curves, with an emphasis on important parameters and their relation to hashing to curves. For further reference on elliptic curves, consult [CFADLNV05] or [W08].

Let F be the finite field GF(q) of prime characteristic p > 3. (This document does not consider elliptic curves over fields of characteristic 2 or 3.) In most cases F is a prime field, so q = p. Otherwise, F is an extension field, so q = p^m for an integer m > 1. This document writes elements of extension fields in a primitive element or polynomial basis, i.e., as a vector of m elements of GF(p) written in ascending order by degree. The entries of this vector are indexed in ascending order starting from 1, i.e., x = (x_1, x_2, ..., x_m). For example, if q = p^2 and the primitive element basis is (1, I), then x = (a, b) corresponds to the element a + b * I, where x_1 = a and x_2 = b.

An elliptic curve E is specified by an equation in two variables and a finite field F. An elliptic curve equation takes one of several standard forms, including (but not limited to) Weierstrass, Montgomery, and Edwards.

The curve E induces an algebraic group whose elements are those points with coordinates (x, y) satisfying the curve equation, and where x and y are elements of F. This group has order n, meaning that there are n distinct points. This document uses additive notation for the elliptic curve group operation.

For security reasons, groups of prime order MUST be used. Elliptic curves induce subgroups of prime order. Let G be a subgroup of the curve of prime order r, where n = h * r. In this equation, h is an integer called the cofactor. An algorithm that takes as input an arbitrary point on the curve E and produces as output a point in the subgroup G of E is said to "clear the cofactor." Such algorithms are discussed in Section 7.

Certain hash-to-curve algorithms restrict the form of the curve equation, the characteristic of the field, and/or the parameters of the curve. For each algorithm presented, this document lists the relevant restrictions.

Summary of quantities:

Symbol	Meaning	Relevance
F,q,p	Finite field F of characteristic p and #F = q = p^m.	For prime fields, q = p; otherwise, q = p^m and m>1.
E	Elliptic curve.	E is specified by an equation and a field F.
n	Number of points on the elliptic curve E.	n = h * r, for h and r defined below.
G	A subgroup of the elliptic curve.	Destination group to which byte strings are encoded.
r	Order of G.	This number MUST be prime.
h	Cofactor, h >= 1.	An integer satisfying n = h * r.

Table 1

2.2. Terminology

In this section, we define important terms used in the rest of this document.

2.2.1. Mappings

A mapping is a deterministic function from an element of the field F to a point on an elliptic curve E defined over F.

In general, the set of all points that a mapping can produce over all possible inputs may be only a subset of the points on an elliptic curve (i.e., the mapping may not be surjective). In addition, a mapping may output the same point for two or more distinct inputs (i.e., the mapping may not be injective). For example, consider a mapping from F to an elliptic curve having n points: if the number of elements of F is not equal to n, then this mapping cannot be bijective (i.e., both injective and surjective) since it is defined to be deterministic.

Mappings may also be invertible, meaning that there is an efficient algorithm that, for any point P output by the mapping, outputs an x in F such that applying the mapping to x outputs P. Some of the mappings given in Section 6 are invertible, but this document does not discuss inversion algorithms.

2.2.2. Encodings

Encodings are closely related to mappings. Like a mapping, an encoding is a function that outputs a point on an elliptic curve. In contrast to a mapping, however, the input to an encoding is an arbitrary string. Encodings can be deterministic or probabilistic. Deterministic encodings are preferred for security, because probabilistic ones are more likely to leak information through side channels.

This document constructs deterministic encodings by composing a hash function H with a deterministic mapping. In particular, H takes as input an arbitrary string and outputs an element of F. The deterministic mapping takes that element as input and outputs a point on an elliptic curve E defined over F. Since the hash function H takes arbitrary strings as inputs, it cannot be injective: the set of inputs is larger than the set of outputs, so there must be distinct inputs that give the same output (i.e., there must be collisions). Thus, any encoding built from H is also not injective.

Like mappings, encodings may be invertible, meaning that there is an efficient algorithm that, for any point P output by the encoding, outputs a string s such that applying the encoding to s outputs P. The hash function used by all encodings specified in this document (Section 5) is not invertible; thus, the encodings are also not invertible.

2.2.3. Random oracle encodings

Two different types of encodings are possible: nonuniform encodings, whose output distribution is not uniformly random, and random oracle encodings, whose output distribution is indistinguishable from uniformly random. Some protocols require a random oracle for security, while others can be securely instantiated with a nonuniform encoding. When the required encoding is not clear, applications SHOULD use a random oracle.

Care is required when constructing a random oracle from a mapping function. A simple but insecure approach is to use the output of a cryptographically secure hash function H as the input to the mapping. Because in general the mapping is not surjective, the output of this construction is distinguishable from uniformly random, i.e., it does not behave like a random oracle.

Brier et al. [BCIMRT10] describe two generic methods for constructing random oracle encodings. Farashahi et al. [FFSTV13] and Tibouchi and Kim [TK17] refine the analysis of one of these constructions. That construction is described in Section 3.

(In more detail: both constructions are indifferentiable from a random oracle [MRH04] when instantiated with appropriate hash functions modeled as random oracles. See Section 10 for further discussion.)

2.2.4. Serialization

A procedure related to encoding is the conversion of an elliptic curve point to a bit string. This is called serialization, and is typically used for compactly storing or transmitting points. The reverse operation, deserialization, converts a bit string to an elliptic curve point. For example, [SEC1] and [p1363a] give standard methods for serialization and deserialization.

Deserialization is different from encoding in that only certain strings (namely, those output by the serialization procedure) can be deserialized. In contrast, this document is concerned with encodings from arbitrary strings to elliptic curve points. This document does not cover serialization or deserialization.

2.2.5. Domain separation

Cryptographic protocols that use random oracles are often analyzed under the assumption that random oracles answer only queries generated by that protocol. In practice, this assumption does not hold if two protocols query the same random oracle. Concretely, consider protocols P1 and P2 that query random oracle RO: if P1 and P2 both query RO on the same value x, the security analysis of one or both protocols may be invalidated.

A common approach to addressing this issue is called domain separation, which allows a single random oracle to simulate multiple, independent oracles. This is effected by ensuring that each simulated oracle sees queries that are distinct from those seen by all other simulated oracles. For example, to simulate two oracles RO1 and RO2 given a single oracle RO, one might define

```
RO1(x) := RO("RO1" || x)
RO2(x) := RO("RO2" || x)
```

In this example, "RO1" and "RO2" are called domain separation tags; they ensure that queries to RO1 and RO2 cannot result in identical queries to RO. Thus, it is safe to treat RO1 and RO2 as independent oracles.

3. Roadmap

This section presents a general framework for encoding byte strings to points on an elliptic curve. To construct these encodings, we rely on three basic functions:

- The function hash_to_field, {0, 1}^* x {0, 1, 2} -> F, hashes arbitrary-length byte strings to elements of a finite field; its implementation is defined in Section 5.

- The function map_to_curve, F -> E, calculates a point on the elliptic curve E from an element of the finite field F over which E is defined. Section 6 describes mappings for a range of curve families.

- The function clear_cofactor, E -> G, sends any point on the curve E to the subgroup G of E. Section 7 describes methods to perform this operation.

We describe two high-level encoding functions (Section 2.2.2). Although these functions have the same interface, the distributions of their outputs are different.

- Nonuniform encoding (encode_to_curve). This function encodes byte strings to points in G. The distribution of the output is not uniformly random in G.

```
encode_to_curve(msg)

Input: msg, an arbitrary-length byte string.
Output: P, a point in G.

Steps:
1. u = hash_to_field(msg, 1)
2. Q = map_to_curve(u[0])
3. P = clear_cofactor(Q)
4. return P
```

- Random oracle encoding (hash_to_curve). This function encodes byte strings to points in G. This function is suitable for applications requiring a random oracle returning points in G, provided that map_to_curve is "well distributed" ([FFSTV13], Def. 1). All of the map_to_curve functions defined in Section 6 meet this requirement.

```
hash_to_curve(msg)

Input: msg, an arbitrary-length byte string.
Output: P, a point in G.

Steps:
1. u = hash_to_field(msg, 2)
2. Q0 = map_to_curve(u[0])
3. Q1 = map_to_curve(u[1])
4. R = Q0 + Q1              # Point addition
5. P = clear_cofactor(R)
6. return P
```

Instances of these functions are given in Section 8, which defines a list of suites that specify a full set of parameters matching elliptic curves and algorithms.

3.1. Domain separation requirements

All uses of the encoding functions defined in this document MUST include domain separation (Section 2.2.5) to avoid interfering with other uses of similar functionality.

Protocols that instantiate multiple, independent hash functions based on either hash_to_curve or encode_to_curve MUST enforce domain separation between those hash functions. This requirement applies both in the case of multiple hashes to the same curve and in the case of multiple hashes to different curves. (This is because the hash_to_field primitive (Section 5) requires domain separation to guarantee independent outputs.)

Domain separation is enforced with a domain separation tag (DST), which is a byte string. Care is required when selecting and using a domain separation tag. The following requirements apply:

1. Tags MUST be supplied as the DST parameter to hash_to_field, as described in Section 5.

2. Tags MUST begin with a fixed protocol identification string. This identification string should be unique to the protocol.

3. Tags SHOULD include a protocol version number.

4. For protocols that define multiple ciphersuites, each ciphersuite's tag MUST be different. For this purpose, it is RECOMMENDED to include a ciphersuite identifier in each tag.

5. For protocols that use multiple encodings, either to the same curve or to different curves, each encoding MUST use a different tag. For this purpose, it is RECOMMENDED to include

the encoding's Suite ID (Section 8) in the domain separation tag. For independent encodings based on the same suite, each tag should also include a distinct identifier, e.g., "ENC1" and "ENC2".

As an example, consider a fictional protocol named Quux that defines several different ciphersuites. A reasonable choice of tag is "QUUX-V<xx>-CS<yy>", where <xx> and <yy> are two-digit numbers indicating the version and ciphersuite, respectively.

As another example, consider a fictional protocol named Baz that requires two independent random oracles, where one oracle outputs points on the curve E1 and the other outputs points on the curve E2. Reasonable choices of tags for the E1 and E2 oracles are "BAZ-V<xx>-CS<yy>-E1" and "BAZ-V<xx>-CS<yy>-E2", respectively, where <xx> and <yy> are as described above.

4. Utility Functions

Algorithms in this document make use of utility functions described below.

For security reasons, all field operations, comparisons, and assignments MUST be implemented in constant time (i.e., execution time MUST NOT depend on the values of the inputs), and without branching. Guidance on implementing these low-level operations in constant time is beyond the scope of this document.

- CMOV(a, b, c): If c is False, CMOV returns a, otherwise it returns b. To prevent against timing attacks, this operation must run in constant time, without revealing the value of c. Commonly, implementations assume that the selector c is 1 for True or 0 for False. In this case, given a bit string C, the desired selector c can be computed by OR-ing all bits of C together. The resulting selector will be either 0 if all bits of C are zero, or 1 if at least one bit of C is 1.

- is_square(x): This function returns True whenever the value x is a square in the field F. Due to Euler's criterion, this function can be calculated in constant time as

```
is_square(x) := { True,  if x^((q - 1) / 2) is 0 or 1 in F;
                { False, otherwise.
```

- sqrt(x): The sqrt operation is a multi-valued function, i.e. there exist two roots of x in the field F whenever x is square. To maintain compatibility across implementations while allowing implementors leeway for optimizations, this document does not require sqrt() to return a particular value. Instead, as explained in Section 6.4, any higher-level function that computes

square roots also specifies how to determine the sign of the result.

The preferred way of computing square roots is to fix a deterministic algorithm particular to F. We give several algorithms in Appendix F. Regardless of the method chosen, the sqrt function should be implemented in a way that resists timing side channels, i.e., in constant time.

* sgn0(x): This function returns either +1 or -1 indicating the "sign" of x, where sgn0(x) == -1 just when x is "negative". In other words, this function always considers 0 to be positive. This function may be implemented in multiple ways; Section 4.1 defines two variants. Throughout the document, sgn0 is used generically to mean either of these variants. Each suite in Section 8 specifies the sgn0 variant to be used.

* inv0(x): This function returns the multiplicative inverse of x in F, extended to all of F by fixing inv0(0) == 0. To implement inv0 in constant time, compute inv0(x) := x^(q - 2). Notice on input 0, the output is 0 as required.

* I2OSP and OS2IP: These functions are used to convert a byte string to and from a non-negative integer as described in [RFC8017].

* a || b: denotes the concatenation of strings a and b.

* substr(str, sstart, slen): for a byte string str, this function returns the slen-byte substring starting at position sstart; positions are zero indexed. For example, substr("ABCDEFG", 2, 3) == "CDE".

* len(str): for a byte string str, this function returns the length of str in bytes. For example, len("ABC") == 3.

* strxor(str1, str2): for byte strings str1 and str2, strxor(str1, str2) returns the bitwise XOR of the two strings. For example, strxor("abc", "XYZ") == "9;9" (the strings in this example are ASCII literals, but strxor is defined for arbitrary byte strings). In this document, strxor is only applied to inputs of equal length.

4.1. sgn0 variants

This section defines two ways of determining the "sign" of an element of F. The variant that should be used is a matter of convention. Other sgn0 variants are possible, but the two given below cover commonly used notions of sign.

It is RECOMMENDED to select the variant that matches the point decompression method of the target curve. In particular, since point decompression requires computing a square root and then choosing the sign of the resulting point, all decompression methods specify, implicitly or explicitly, a method for determining the sign of an element of F. It is convenient for hash-to-curve and decompression to agree on a notion of sign, since this may permit simpler implementations.

See Section 2.1 for a discussion of representing elements of extension fields as vectors; this representation is used in both of the sgn0 variants below.

Note that any valid sgn0 function for extension fields must iterate over the entire vector representation of the input element. To see why, imagine a function sgn0* that ignores the final entry in its input vector, and consider a field element x = (0, x_2). Since sgn0* ignores x_2, sgn0*(x) == sgn0*(-x), which is incorrect when x_2 != 0. The same argument applies to all entries of any x, establishing the claim.

4.1.1. Big endian variant

The following sgn0 variant is defined such that sgn0_be(x) = -1 just when the big-endian encoding of x is lexically greater than the encoding of -x.

This variant SHOULD be used when points on the target elliptic curve are serialized using the SORT compression method given in IEEE 1363a-2004 [p1363a], Section 5.5.6.1.2, and other similar methods.

```
sgn0_be(x)

Parameters:
- F, a finite field of characteristic p and order q = p^m.
- p, the characteristic of F (see immediately above).
- m, the extension degree of F, m >= 1 (see immediately above).

Input: x, an element of F.
Output: -1 or 1 (an integer).

Notation: x_i is the i^th element of the vector representation of x.

Steps:
1. sign = 0
2. for i in (m, m - 1, ..., 1):
3.   sign_i = CMOV(1, -1, x_i > ((p - 1) / 2))
4.   sign_i = CMOV(sign_i, 0, x_i == 0)
5.   sign = CMOV(sign, sign_i, sign == 0)
6. return CMOV(sign, 1, sign == 0)      # Regard x == 0 as positive
```

4.1.2. Little endian variant

The following sgn0 variant is defined such that sgn0_le(x) = -1 just when x != 0 and the parity of the least significant nonzero entry of the vector representation of x is 1.

This variant SHOULD be used when points on the target elliptic curve are serialized using any of the following methods:

- the LSB compression method given in IEEE 1363a-2004 [p1363a], Section 5.5.6.1.1,
- the method given in [SEC1] Section 2.3.3, or
- the method given in ANSI X9.62-1998 [x9.62], Section 4.2.1.

This variant is also compatible with the compression method specified for the Ed25519 and Ed448 elliptic curves [RFC8032].

```
sgn0_le(x)

Parameters:
- F, a finite field of characteristic p and order q = p^m.
- p, the characteristic of F (see immediately above).
- m, the extension degree of F, m >= 1 (see immediately above).

Input: x, an element of F.
Output: -1 or 1 (an integer).

Notation: x_i is the i^th element of the vector representation of x.

Steps:
1. sign = 0
2. for i in (1, 2, ..., m):
3.   sign_i = CMOV(1, -1, x_i mod 2 == 1)
4.   sign_i = CMOV(sign_i, 0, x_i == 0)
5.   sign = CMOV(sign, sign_i, sign == 0)
6. return CMOV(sign, 1, sign == 0)      # Regard x == 0 as positive
```

5. Hashing to a Finite Field

The hash_to_field function hashes a byte string msg of any length into one or more elements of a field F. This function works in two steps: it first hashes the input byte string to produce a pseudorandom byte string, and then interprets this pseudorandom byte string as one or more elements of F.

For the first step, hash_to_field calls an auxiliary function expand_message. This document defines two variants of expand_message, one appropriate for hash functions like SHA-2

[FIPS180-4] or SHA-3 [FIPS202], and one appropriate for extensible-output functions like SHAKE-128 [FIPS202]. Security considerations for each expand_message variant are discussed below (Section 5.3.1, Section 5.3.2).

Implementors MUST NOT use rejection sampling to generate a uniformly random element of F. The reason is that rejection sampling procedures are difficult to implement in constant time, and later well-meaning "optimizations" may silently render an implementation non-constant-time.

5.1. Security considerations

The hash_to_field function is designed to be indifferentiable from a random oracle [MRH04] when expand_message (Section 5.3) is modeled as a random oracle (see Section 10.1). Ensuring indifferentiability requires care; to see why, consider a prime p that is close to 3/4 * 2^256. Reducing a random 256-bit integer modulo this p yields a value that is in the range [0, p / 3] with probability roughly 1/2, meaning that this value is statistically far from uniform in [0, p - 1].

To control bias, hash_to_field instead uses pseudorandom integers whose length is at least ceil(log2(p)) + k bits. Reducing such integers mod p gives bias at most 2^-k for any p; this bias is appropriate when targeting k-bit security. To obtain such integers, hash_to_field uses expand_message to obtain L pseudorandom bytes, where L = ceil((ceil(log2(p)) + k) / 8); this byte string is then interpreted as an integer via OS2IP [RFC8017]. For example, for a 255-bit prime p, and k = 128-bit security, L = ceil((255 + 128) / 8) = 48 bytes.

5.2. hash_to_field implementation

The following procedure implements hash_to_field.

The expand_message parameter to this function MUST conform to the requirements given below (Section 5.3).

Section 3.1 discusses requirements for domain separation and recommendations for choosing DST, the domain separation tag. This is the REQUIRED method for applying domain separation.

```
hash_to_field(msg, count)

Parameters:
- DST, a domain separation tag (see discussion above).
- F, a finite field of characteristic p and order q = p^m.
- p, the characteristic of F (see immediately above).
- m, the extension degree of F, m >= 1 (see immediately above).
- L = ceil((ceil(log2(p)) + k) / 8), where k is the security
  parameter of the cryptosystem (e.g., k = 128).
- expand_message, a function that expands a byte string and
  domain separation tag into a pseudorandom byte string
  (see discussion above).

Inputs:
- msg is a byte string containing the message to hash.
- count is the number of elements of F to output.

Outputs:
- (u_0, ..., u_(count - 1)), a list of field elements.

Steps:
1. len_in_bytes = count * m * L
2. pseudo_random_bytes = expand_message(msg, DST, len_in_bytes)
3. for i in (0, ..., count - 1):
4.   for j in (0, ..., m - 1):
5.     elm_offset = L * (j + i * m)
6.     tv = substr(pseudo_random_bytes, elm_offset, L)
7.     e_j = OS2IP(tv) mod p
8.   u_i = (e_0, ..., e_(m - 1))
9. return (u_0, ..., u_(count - 1))
```

5.3. expand_message

expand_message is a function that generates a pseudorandom byte string. It takes three arguments:

- *msg, a byte string containing the message to hash,

- *DST, a byte string that acts as a domain separation tag, and

- *len_in_bytes, the number of bytes to be generated.

This document defines two variants of expand_message:

- *expand_message_xmd (Section 5.3.1) is appropriate for use with a wide range of hash functions, including SHA-2 [FIPS180-4], SHA-3 [FIPS202], BLAKE2 [RFC7693], and others.

- *expand_message_xof (Section 5.3.2) is appropriate for use with extensible-output functions (XOFs) including functions in the SHAKE [FIPS202] or BLAKE2X [BLAKE2X] families.

These variants should suffice for the vast majority of use cases, but other variants are possible; Section 5.3.3 discusses requirements.

The expand_message variants defined in this section accept domain separation tags of at most 255 bytes. If a domain separation tag longer than 255 bytes must be used (e.g., because of requirements imposed by an invoking protocol), implementors MUST compute a short domain separation tag by hashing, as follows:

```
DST = H("H2C-OVERSIZE-DST-" || a_very_long_DST)
```

Here, a_very_long_DST is the DST whose length is greater than 255 bytes, "H2C-OVERSIZE-DST-" is an ASCII string literal, and the hash function H MUST meet the criteria given in Section 5.3.1.

5.3.1. expand_message_xmd

The expand_message_xmd function produces a pseudorandom byte string using a cryptographic hash function H that outputs b bits. For security, H must meet the following requirements:

- *The number of bits output by H MUST be b >= 2 * k, for k the target security level in bits. This ensures k-bit collision resistance.

- *H MAY be a Merkle-Damgaard hash function like SHA-2. In this case, security holds when the underlying compression function is modeled as a random oracle [CDMP05]. (See Section 10.2 for discussion.)

- *H MAY be a sponge-based hash function like SHA-3 or BLAKE2. In this case, security holds when the inner function is modeled as a random transformation or as a random permutation [BDPV08].

- *Otherwise, H MUST be a hash function that has been proved indifferentiable from a random oracle [MRH04] under a widely accepted cryptographic assumption.

SHA-2 [FIPS180-4] and SHA-3 [FIPS202] are typical and RECOMMENDED choices. As an example, for the 128-bit security level, b >= 256 bits and either SHA-256 or SHA3-256 would be an appropriate choice.

The following procedure implements expand_message_xmd.

```
expand_message_xmd(msg, DST, len_in_bytes)

Parameters:
- H, a hash function (see requirements above).
- b_in_bytes, ceil(b / 8) for b the output size of H in bits.
  For example, for b = 256, b_in_bytes = 32.
- r_in_bytes, the input block size of H, measured in bytes.
  For example, for SHA-256, r_in_bytes = 64.

Input:
- msg, a byte string.
- DST, a byte string of at most 255 bytes.
- len_in_bytes, the length of the requested output in bytes.

Output:
- pseudo_random_bytes, a byte string

Steps:
1.  ell = ceil(len_in_bytes / b_in_bytes)
2.  ABORT if ell > 255
3.  DST_prime = I2OSP(len(DST), 1) || DST
4.  Z_pad = I2OSP(0, r_in_bytes)
5.  l_i_b_str = I2OSP(len_in_bytes, 2)
6.  b_0 = H(Z_pad || msg || l_i_b_str || I2OSP(0, 1) || DST_prime)
7.  b_1 = H(b_0 || I2OSP(1, 1) || DST_prime)
8.  for i in (2, ..., ell):
9.    b_i = H(strxor(b_0, b_(i - 1)) || I2OSP(i, 1) || DST_prime)
10. pseudo_random_bytes = b_1 || ... || b_ell
11. return substr(pseudo_random_bytes, 0, len_in_bytes)
```

Note that the string Z_pad is prepended to msg when computing b_0 (step 6). This is necessary for security when H is a Merkle-Damgaard hash, e.g., SHA-2 (see Section 10.2). Hashing this additional data means that the cost of computing b_0 is higher than the cost of simply computing H(msg). In most settings this overhead is negligible, because the cost of evaluating H is much less than the other costs involved in hashing to a curve.

It is possible, however, to entirely avoid this overhead by taking advantage of the fact that Z_pad depends only on H, and not on the arguments to expand_message_xmd. To do so, first precompute and save the internal state of H after ingesting Z_pad; and then, when computing b_0, initialize H using the saved state. Further details are beyond the scope of this document.

5.3.2. expand_message_xof

The expand_message_xof function produces a pseudorandom byte string using an extensible-output function (XOF) H. For security, H must meet the following criteria:

- The collision resistance of H MUST be at least k bits.
- H MUST be an XOF that has been proved indifferentiable from a random oracle under a reasonable cryptographic assumption.

The SHAKE [FIPS202] XOF family is a typical and RECOMMENDED choice. As an example, for 128-bit security, SHAKE-128 would be an appropriate choice.

The following procedure implements expand_message_xof.

```
expand_message_xof(msg, DST, len_in_bytes)

Parameters:
- H, an extensible-output function.
  H(m, d) hashes message m and returns d bytes.

Input:
- msg, a byte string.
- DST, a byte string of at most 255 bytes.
- len_in_bytes, the length of the requested output in bytes.

Output:
- pseudo_random_bytes, a byte string

Steps:
1. DST_prime = I2OSP(len(DST), 1) || DST
2. msg_prime = msg || I2OSP(len_in_bytes, 2) || DST_prime
3. pseudo_random_bytes = H(msg_prime, len_in_bytes)
4. return pseudo_random_bytes
```

5.3.3. Defining other expand_message variants

When defining a new expand_message variant, the most important consideration is that hash_to_field models expand_message as a random oracle. Thus, implementors SHOULD prove indifferentiability from a random oracle under an appropriate assumption about the underlying cryptographic primitives.

In addition, expand_message variants:

- MUST give collision resistance commensurate with the security level of the target elliptic curve.

- *MUST be built on primitives designed for use in applications requiring cryptographic randomness. As examples, a secure stream cipher is an appropriate primitive, whereas a Mersenne twister pseudorandom number generator is not.

- *MUST NOT use any form of rejection sampling.

- *MUST give independent values for distinct (msg, DST, length) inputs. Meeting this requirement is slightly subtle. As a simplified example, hashing the concatenation msg || DST does not work, because in this case distinct (msg, DST) pairs whose concatenations are equal will return the same output (e.g., ("AB", "CDEF") and ("ABC", "DEF")). The variants defined in this document use a prefix-free encoding of DST to avoid this issue.

- *MUST use the domain separation tag DST to ensure that invocations of cryptographic primitives inside of expand_message are domain separated from invocations outside of expand_message. For example, if the expand_message variant uses a hash function H, an encoding of DST MUST be either prepended or appended to the input to each invocation of H (appending is the RECOMMENDED approach).

- *SHOULD read msg exactly once, for efficiency when msg is long.

In addition, an expand_message variant MUST specify a unique EXP_TAG that identifies that variant in a Suite ID. See Section 8.9 for more information.

6. Deterministic Mappings

The mappings in this section are suitable for constructing either nonuniform or random oracle encodings using the constructions of Section 3. Certain mappings restrict the form of the curve or its parameters. For each mapping presented, this document lists the relevant restrictions.

Note that mappings in this section are not interchangeable: different mappings will almost certainly output different points when evaluated on the same input.

6.1. Choosing a mapping function

This section gives brief guidelines on choosing a mapping function for a given elliptic curve. Note that the suites given in Section 8 are recommended mappings for the respective curves.

If the target elliptic curve is a Montgomery curve (Section 6.7), the Elligator 2 method (Section 6.7.1) is recommended. Similarly, if the target elliptic curve is a twisted Edwards curve (Section 6.8),

the twisted Edwards Elligator 2 method (Section 6.8.2) is recommended.

The remaining cases are Weierstrass curves. For curves supported by the Simplified SWU method (Section 6.6.2), that mapping is the recommended one. Otherwise, the Simplified SWU method for AB == 0 (Section 6.6.3) is recommended if the goal is best performance, while the Shallue-van de Woestijne method (Section 6.6.1) is recommended if the goal is simplicity of implementation. (The reason for this distinction is that the Simplified SWU method for AB == 0 requires implementing an isogeny map in addition to the mapping function, while the Shallue-van de Woestijne method does not.)

The Shallue-van de Woestijne method (Section 6.6.1) works with any curve, and may be used in cases where a generic mapping is required. Note, however, that this mapping is almost always more computationally expensive than the curve-specific recommendations above.

6.2. Interface

The generic interface shared by all mappings in this section is as follows:

```
(x, y) = map_to_curve(u)
```

The input u and outputs x and y are elements of the field F. The coordinates (x, y) specify a point on an elliptic curve defined over F. Note that the point (x, y) is not a uniformly random point. If uniformity is required for security, the random oracle construction of Section 3 MUST be used instead.

6.3. Notation

As a rough guide, the following conventions are used in pseudocode:

- All arithmetic operations are performed over a field F, unless explicitly stated otherwise.
- u: the input to the mapping function. This is an element of F produced by the hash_to_field function.
- (x, y), (s, t), (v, w): the affine coordinates of the point output by the mapping. Indexed variables (e.g., x1, y2, ...) are used for candidate values.
- tv1, tv2, ...: reusable temporary variables.
- c1, c2, ...: constant values, which can be computed in advance.

6.4. Sign of the resulting point

In general, elliptic curves have equations of the form y^2 = g(x). Most of the mappings in this section first identify an x such that g(x) is square, then take a square root to find y. Since there are two square roots when g(x) != 0, this results in an ambiguity regarding the sign of y.

To resolve this ambiguity, the mappings in this section specify the sign of the y-coordinate in terms of the input to the mapping function. Two main reasons support this approach. First, this covers elliptic curves over any field in a uniform way, and second, it gives implementors leeway to optimize their square-root implementations.

6.5. Exceptional cases

Mappings may have have exceptional cases, i.e., inputs u on which the mapping is undefined. These cases must be handled carefully, especially for constant-time implementations.

For each mapping in this section, we discuss the exceptional cases and show how to handle them in constant time. Note that all implementations SHOULD use inv0 (Section 4) to compute multiplicative inverses, to avoid exceptional cases that result from attempting to compute the inverse of 0.

6.6. Mappings for Weierstrass curves

The mappings in this section apply to a target curve E defined by the equation

```
y^2 = g(x) = x^3 + A * x + B
```

where 4 * A^3 + 27 * B^2 != 0.

6.6.1. Shallue-van de Woestijne Method

Shallue and van de Woestijne [SW06] describe a mapping that applies to essentially any elliptic curve. (Note, however, that this mapping is more expensive to evaluate than the other mappings in this document.)

The parameterization given below is for Weierstrass curves; its derivation is detailed in [W19]. This parameterization also works for Montgomery (Section 6.7) and twisted Edwards (Section 6.8) curves via the rational maps given in Appendix B: first evaluate the Shallue-van de Woestijne mapping to an equivalent Weierstrass curve, then map that point to the target Montgomery or twisted Edwards curve using the corresponding rational map.

Preconditions: A Weierstrass curve y^2 = x^3 + A * x + B.

Constants:

- *A and B, the parameter of the Weierstrass curve.

- *Z, an element of F meeting the below criteria. Appendix E.1 gives a Sage [SAGE] script that outputs the RECOMMENDED Z.

  1. g(Z) != 0 in F.

  2. -(3 * Z^2 + 4 * A) / (4 * g(Z)) != 0 in F.

  3. -(3 * Z^2 + 4 * A) / (4 * g(Z)) is square in F.

  4. At least one of g(Z) and g(-Z / 2) is square in F.

Sign of y: Inputs u and -u give the same x-coordinate. Thus, we set sgn0(y) == sgn0(u).

Exceptions: The exceptional cases for u occur when (1 + u^2 * g(Z)) * (1 - u^2 * g(Z)) == 0. The restrictions on Z given above ensure that implementations that use inv0 to invert this product are exception free.

Operations:

```
1. tv1 = u^2 * g(Z)
2. tv2 = 1 + tv1
3. tv1 = 1 - tv1
4. tv3 = inv0(tv1 * tv2)
5. tv4 = sqrt(-g(Z) * (3 * Z^2 + 4 * A))
6. tv4 = tv4 * sgn0(tv4)                    # sgn0(tv4) MUST equal 1
7. tv5 = u * tv1 * tv3 * tv4
8.  x1 = -Z / 2 - tv5
9.  x2 = -Z / 2 + tv5
10. x3 = Z - 4 * g(Z) * (tv2^2 * tv3)^2 / (3 * Z^2 + 4 * A)
11. If is_square(g(x1)), set x = x1 and y = sqrt(g(x1))
12. Else If is_square(g(x2)), set x = x2 and y = sqrt(g(x2))
13. Else set x = x3 and y = sqrt(g(x3))
14. If sgn0(u) != sgn0(y), set y = -y
15. return (x, y)
```

6.6.1.1. Implementation

The following procedure implements the Shallue and van de Woestijne method in a straight-line fashion.

```
map_to_curve_svdw(u)
Input: u, an element of F.
Output: (x, y), a point on E.

Constants:
1. c1 = g(Z)
2. c2 = -Z / 2
3. c3 = sqrt(-g(Z) * (3 * Z^2 + 4 * A))     # sgn0(c3) MUST equal 1
4. c4 = -4 * g(Z) / (3 * Z^2 + 4 * A)

Steps:
1.  tv1 = u^2
2.  tv1 = tv1 * c1
3.  tv2 = 1 + tv1
4.  tv1 = 1 - tv1
5.  tv3 = tv1 * tv2
6.  tv3 = inv0(tv3)
7.  tv4 = u * tv1
8.  tv4 = tv4 * tv3
9.  tv4 = tv4 * c3
10.  x1 = c2 - tv4
11. gx1 = x1^2
12. gx1 = gx1 + A
13. gx1 = gx1 * x1
14. gx1 = gx1 + B
15.  e1 = is_square(gx1)
16.  x2 = c2 + tv4
17. gx2 = x2^2
18. gx2 = gx2 + A
19. gx2 = gx2 * x2
20. gx2 = gx2 + B
21.  e2 = is_square(gx2) AND NOT e1     # Avoid short-circuit logic ops
22.  x3 = tv2^2
23.  x3 = x3 * tv3
24.  x3 = x3^2
25.  x3 = x3 * c4
26.  x3 = x3 + Z
27.   x = CMOV(x3, x1, e1)      # x = x1 if gx1 is square, else x = x3
28.   x = CMOV(x, x2, e2)       # x = x2 if gx2 is square and gx1 is not
29.  gx = x^2
30.  gx = gx + A
31.  gx = gx * x
32.  gx = gx + B
33.   y = sqrt(gx)
34.  e3 = sgn0(u) == sgn0(y)
35.   y = CMOV(-y, y, e3)       # Select correct sign of y
36. return (x, y)
```

6.6.2. Simplified Shallue-van de Woestijne-Ulas Method

The function map_to_curve_simple_swu(u) implements a simplification of the Shallue-van de Woestijne-Ulas mapping [U07] described by Brier et al. [BCIMRT10], which they call the "simplified SWU" map. Wahby and Boneh [WB19] generalize and optimize this mapping.

Preconditions: A Weierstrass curve y^2 = x^3 + A * x + B where A != 0 and B != 0.

Constants:

- *A and B, the parameters of the Weierstrass curve.
- *Z, an element of F meeting the below criteria. Appendix E.2 gives a Sage [SAGE] script that outputs the RECOMMENDED Z. The criteria are:
  1. Z is non-square in F,
  2. Z != -1 in F,
  3. the polynomial g(x) - Z is irreducible over F, and
  4. g(B / (Z * A)) is square in F.

Sign of y: Inputs u and -u give the same x-coordinate. Thus, we set sgn0(y) == sgn0(u).

Exceptions: The exceptional cases are values of u such that Z^2 * u^4 + Z * u^2 == 0. This includes u == 0, and may include other values depending on Z. Implementations must detect this case and set x1 = B / (Z * A), which guarantees that g(x1) is square by the condition on Z given above.

Operations:

```
1. tv1 = inv0(Z^2 * u^4 + Z * u^2)
2.  x1 = (-B / A) * (1 + tv1)
3.  If tv1 == 0, set x1 = B / (Z * A)
4. gx1 = x1^3 + A * x1 + B
5.  x2 = Z * u^2 * x1
6. gx2 = x2^3 + A * x2 + B
7.  If is_square(gx1), set x = x1 and y = sqrt(gx1)
8.  Else set x = x2 and y = sqrt(gx2)
9.  If sgn0(u) != sgn0(y), set y = -y
10. return (x, y)
```

6.6.2.1. Implementation

The following procedure implements the simplified SWU mapping in a straight-line fashion. Appendix D gives an optimized straight-line procedure for P-256 [FIPS186-4]. For more information on optimizing this mapping, see [WB19] Section 4 or the example code found at [hash2curve-repo].

```
map_to_curve_simple_swu(u)
Input: u, an element of F.
Output: (x, y), a point on E.

Constants:
1.  c1 = -B / A
2.  c2 = -1 / Z

Steps:
1.  tv1 = Z * u^2
2.  tv2 = tv1^2
3.   x1 = tv1 + tv2
4.   x1 = inv0(x1)
5.   e1 = x1 == 0
6.   x1 = x1 + 1
7.   x1 = CMOV(x1, c2, e1)    # If (tv1 + tv2) == 0, set x1 = -1 / Z
8.   x1 = x1 * c1      # x1 = (-B / A) * (1 + (1 / (Z^2 * u^4 + Z * u^2)
9.  gx1 = x1^2
10. gx1 = gx1 + A
11. gx1 = gx1 * x1
12. gx1 = gx1 + B             # gx1 = g(x1) = x1^3 + A * x1 + B
13.  x2 = tv1 * x1            # x2 = Z * u^2 * x1
14. tv2 = tv1 * tv2
15. gx2 = gx1 * tv2           # gx2 = (Z * u^2)^3 * gx1
16.  e2 = is_square(gx1)
17.   x = CMOV(x2, x1, e2)    # If is_square(gx1), x = x1, else x = x2
18.  y2 = CMOV(gx2, gx1, e2)  # If is_square(gx1), y2 = gx1, else y2 = g
19.   y = sqrt(y2)
20.  e3 = sgn0(u) == sgn0(y)  # Fix sign of y
21.   y = CMOV(-y, y, e3)
22. return (x, y)
```

6.6.3. Simplified SWU for AB == 0

Wahby and Boneh [WB19] show how to adapt the simplified SWU mapping to Weierstrass curves having A == 0 or B == 0, which the mapping of Section 6.6.2 does not support. (The case A == B == 0 is excluded because y^2 = x^3 is not an elliptic curve.)

This method applies to curves like secp256k1 [SEC2] and to pairing-friendly curves in the Barreto-Lynn-Scott [BLS03], Barreto-Naehrig [BN05], and other families.

This method requires finding another elliptic curve E' given by the equation

```
y'^2 = g'(x') = x'^3 + A' * x' + B'
```

that is isogenous to E and has A' != 0 and B' != 0. (One might do this, for example, using [SAGE]; for details, see [WB19], Appendix A.) This isogeny defines a map iso_map(x', y') that takes as input a point on E' and produces as output a point on E.

Once E' and iso_map are identified, this mapping works as follows: on input u, first apply the simplified SWU mapping to get a point on E', then apply the isogeny map to that point to get a point on E.

Note that iso_map is a group homomorphism, meaning that point addition commutes with iso_map. Thus, when using this mapping in the hash_to_curve construction of Section 3, one can effect a small optimization by first mapping u0 and u1 to E', adding the resulting points on E', and then applying iso_map to the sum. This gives the same result while requiring only one evaluation of iso_map.

Preconditions: An elliptic curve E' with A' != 0 and B' != 0 that is isogenous to the target curve E with isogeny map iso_map from E' to E.

Helper functions:

- *map_to_curve_simple_swu is the mapping of Section 6.6.2 to E'
- *iso_map is the isogeny map from E' to E

Sign of y: for this map, the sign is determined by map_to_curve_simple_swu. No further sign adjustments are necessary.

Exceptions: map_to_curve_simple_swu handles its exceptional cases. Exceptional cases of iso_map MUST return the identity point on E.

Operations:

```
1. (x', y') = map_to_curve_simple_swu(u)    # (x', y') is on E'
2.   (x, y) = iso_map(x', y')               # (x, y) is on E
3. return (x, y)
```

See [hash2curve-repo] or [WB19], Section 4.3 for details on implementing the isogeny map.

6.7. Mappings for Montgomery curves

The mapping defined in this section applies to a target curve M defined by the equation

```
K * t^2 = s^3 + J * s^2 + s
```

6.7.1. Elligator 2 Method

Preconditions: A Montgomery curve K * t^2 = s^3 + J * s^2 + s where J != 0, K != 0, and (J^2 - 4) / K^2 is non-zero and non-square in F.

Constants:

- J and K, the parameters of the elliptic curve.
- Z, a non-square element of F. Appendix E.3 gives a Sage [SAGE] script that outputs the RECOMMENDED Z.

Sign of t: Inputs u and -u give the same s-coordinate. Thus, we set sgn0(t) == sgn0(u).

Exceptions: The exceptional case is Z * u^2 == -1, i.e., 1 + Z * u^2 == 0. Implementations must detect this case and set x1 = -(J / K). Note that this can only happen when q = 3 (mod 4).

Operations:

```
1.  x1 = -(J / K) * inv0(1 + Z * u^2)
2.  If x1 == 0, set x1 = -(J / K)
3. gx1 = x1^3 + (J / K) * x1^2 + x1 / K^2
4.  x2 = -x1 - (J / K)
5. gx2 = x2^3 + (J / K) * x2^2 + x2 / K^2
6.  If is_square(gx1), set x = x1 and y = sqrt(gx1)
7.  Else set x = x2 and y = sqrt(gx2)
8.   s = x * K
9.   t = y * K
10. If sgn0(u) != sgn0(t), set t = -t
11. return (s, t)
```

6.7.1.1. Implementation

The following procedure implements Elligator 2 in a straight-line fashion. Appendix D gives optimized straight-line procedures for curve25519 and curve448 [RFC7748].

```
map_to_curve_elligator2(u)
Input: u, an element of F.
Output: (s, t), a point on M.

Constants:
1.   c1 = J / K
2.   c2 = 1 / K^2

Steps:
1.  tv1 = u^2
2.  tv1 = Z * tv1             # Z * u^2
3.   e1 = tv1 == -1           # exceptional case: Z * u^2 == -1
4.  tv1 = CMOV(tv1, 0, e1)    # if tv1 == -1, set tv1 = 0
5.   x1 = tv1 + 1
6.   x1 = inv0(x1)
7.   x1 = -c1 * x1             # x1 = -(J / K) / (1 + Z * u^2)
8.  gx1 = x1 + c1
9.  gx1 = gx1 * x1
10. gx1 = gx1 + c2
11. gx1 = gx1 * x1            # gx1 = x1^3 + (J / K) * x1^2 + x1 / K^2
12.  x2 = -x1 - c1
13. gx2 = tv1 * gx1
14.  e2 = is_square(gx1)
15.   x = CMOV(x2, x1, e2)    # If is_square(gx1), x = x1, else x = x2
16.  y2 = CMOV(gx2, gx1, e2)  # If is_square(gx1), y2 = gx1, else y2 = g
17.   y = sqrt(y2)
18.   s = x * K
19.   t = y * K
20.  e3 = sgn0(u) == sgn0(t)  # Fix sign of t
21.   t = CMOV(-t, t, e3)
22. return (s, t)
```

6.8. Mappings for Twisted Edwards curves

Twisted Edwards curves (a class of curves that includes Edwards curves) are given by the equation

```
a * v^2 + w^2 = 1 + d * v^2 * w^2
```

with a != 0, d != 0, and a != d [BBJLP08].

These curves are closely related to Montgomery curves (Section 6.7): every twisted Edwards curve is birationally equivalent to a Montgomery curve ([BBJLP08], Theorem 3.2). This equivalence yields an efficient way of hashing to a twisted Edwards curve: first, hash to an equivalent Montgomery curve, then transform the result into a point on the twisted Edwards curve via a rational map. This method of hashing to a twisted Edwards curve thus requires identifying a

corresponding Montgomery curve and rational map. We describe how to identify such a curve and map immediately below.

6.8.1. Rational maps from Montgomery to twisted Edwards curves

There are two ways to identify the correct Montgomery curve and rational map for use when hashing to a given twisted Edwards curve.

When hashing to a standardized twisted Edwards curve for which a corresponding Montgomery form and rational map are also standardized, the standard Montgomery form and rational map MUST be used to ensure compatibility with existing software. Two such standardized curves are the edwards25519 and edwards448 curves, which correspond to the Montgomery curves curve25519 and curve448, respectively. For both of these curves, [RFC7748] lists both the Montgomery and twisted Edwards forms and gives the corresponding rational maps.

The rational map for edwards25519 ([RFC7748], Section 4.1) uses the constant sqrt_neg_486664 = sqrt(-486664) (mod 2^255 - 19). To ensure compatibility, this constant MUST be chosen such that sgn0(sqrt_neg_486664) == 1. Analogous ambiguities in other standardized rational maps MUST be resolved in the same way: for any constant k whose sign is ambiguous, k MUST be chosen such that sgn0(k) == 1.

The 4-isogeny map from curve448 to edwards448 ([RFC7748], Section 4.2) is unambiguous with respect to sign.

When defining new twisted Edwards curves, a Montgomery equivalent and rational map SHOULD be specified, and the sign of the rational map SHOULD be stated unambiguously.

When hashing to a twisted Edwards curve that does not have a standardized Montgomery form or rational map, the following procedure MUST be used to derive them. For a twisted Edwards curve given by

```
a * v^2 + w^2 = 1 + d * v^2 * w^2
```

first compute J and K, the parameters of the equivalent Montgomery curve given by

```
K * t^2 = s^3 + J * s^2 + s
```

as follows:

*J = 2 * (a + d) / (a - d)

*K = 4 / (a - d)

Note that this curve has the form required by the Elligator 2 mapping of Section 6.7.1. The rational map from the point (s, t) on this Montgomery curve to the point (v, w) on the twisted Edwards curve is given by

*v = s / t

*w = (s - 1) / (s + 1)

(For completeness, we give the inverse map in Appendix B.1. Note that the inverse map is not used when hashing to a twisted Edwards curve.)

Rational maps may be undefined on certain inputs, e.g., when the denominator of one of the rational functions is zero. In the map described above, the exceptional cases are t == 0 or s == -1. Implementations MUST detect exceptional cases and return the value (v, w) = (0, 1), which is the identity point on all twisted Edwards curves.

The following straight-line implementation of the above rational map handles the exceptional cases. Implementations of other rational maps (e.g., the ones give in [RFC7748]) are analogous.

```
rational_map(s, t)
Input: (s, t), a point on the curve K * t^2 = s^3 + J * s^2 + s.
Output: (v, w), a point on an equivalent twisted Edwards curve.

1. tv1 = s + 1
2. tv2 = tv1 * t        # (s + 1) * t
3. tv2 = inv0(tv2)      # 1 / ((s + 1) * t)
4.   v = tv2 * tv1      # 1 / t
5.   v = v * s          # s / t
6.   w = tv2 * t        # 1 / (s + 1)
7. tv1 = s - 1
8.   w = w * tv1        # (s - 1) / (s + 1)
9.   e = tv2 == 0
10.  w = CMOV(w, 1, e)  # handle exceptional case
11. return (v, w)
```

6.8.2. Elligator 2 Method

Preconditions: A twisted Edwards curve E and an equivalent Montgomery curve M meeting the requirements in Section 6.8.1.

Helper functions:

*map_to_curve_elligator2 is the mapping of Section 6.7.1 to the curve M.

*rational_map is a function that takes a point (s, t) on M and returns a point (v, w) on E, as defined in Section 6.8.1.

Sign of t (and v): for this map, the sign is determined by map_to_curve_elligator2. No further sign adjustments are required.

Exceptions: The exceptions for the Elligator 2 mapping are as given in Section 6.7.1. The exceptions for the rational map are as given in Section 6.8.1. No other exceptions are possible.

The following procedure implements the Elligator 2 mapping for a twisted Edwards curve. (Note that the output point is denoted (v, w) because it is a point on the target twisted Edwards curve.)

```
map_to_curve_elligator2_edwards(u)
Input: u, an element of F.
Output: (v, w), a point on E.

1. (s, t) = map_to_curve_elligator2(u)      # (s, t) is on M
2. (v, w) = rational_map(s, t)              # (v, w) is on E
3. return (v, w)
```

7. Clearing the cofactor

The mappings of Section 6 always output a point on the elliptic curve, i.e., a point in a group of order h * r (Section 2.1). Obtaining a point in G may require a final operation commonly called "clearing the cofactor," which takes as input any point on the curve.

The cofactor can always be cleared via scalar multiplication by h. For elliptic curves where h = 1, i.e., the curves with a prime number of points, no operation is required. This applies, for example, to the NIST curves P-256, P-384, and P-521 [FIPS186-4].

In some cases, it is possible to clear the cofactor via a faster method than scalar multiplication by h. These methods are equivalent to (but usually faster than) multiplication by some scalar h_eff whose value is determined by the method and the curve. Examples of fast cofactor clearing methods include the following:

*For certain pairing-friendly curves having subgroup G2 over an extension field, Scott et al. [SBCDK09] describe a method for fast cofactor clearing that exploits an efficiently-computable endomorphism. Fuentes-Castaneda et al. [FKR11] propose an alternative method that is sometimes more efficient. Budroni and Pintore [BP17] give concrete instantiations of these methods for Barreto-Lynn-Scott pairing-friendly curves [BLS03].

* Wahby and Boneh ([WB19], Section 5) describe a trick due to Scott for fast cofactor clearing on any elliptic curve for which the prime factorization of h and the structure of the elliptic curve group meet certain conditions.

The clear_cofactor function is parameterized by a scalar h_eff. Specifically,

```
clear_cofactor(P) := h_eff * P
```

where * represents scalar multiplication. When a curve does not support a fast cofactor clearing method, h_eff = h and the cofactor MUST be cleared via scalar multiplication.

When a curve admits a fast cofactor clearing method, clear_cofactor MAY be evaluated either via that method or via scalar multiplication by the equivalent h_eff; these two methods give the same result. Note that in this case scalar multiplication by the cofactor h does not generally give the same result as the fast method, and SHOULD NOT be used.

8. Suites for Hashing

This section lists recommended suites for hashing to standard elliptic curves.

A suite fully specifies the procedure for hashing byte strings to points on a specific elliptic curve group. Each suite comprises the following parameters:

* Suite ID, a short name used to refer to a given suite. Section 8.9 discusses the naming conventions for suite IDs.

* encoding type, either random oracle (hash_to_curve) or nonuniform (encode_to_curve). See Section 3 for definitions of these encoding types.

* E, the target elliptic curve over a field F.

* p, the characteristic of the field F.

* m, the extension degree of the field F.

* k, the target security level of the suite in bits.

* sgn0, one of the variants specified in Section 4.1.

* L, the length parameter for hash_to_field (Section 5.1).

*expand_message, one of the variants specified in Section 5.3 plus any parameters required for the specified variant (for example, H, the underlying hash function).

*f, a mapping function from Section 6.

*h_eff, the scalar parameter for clear_cofactor (Section 7).

In addition to the above parameters, the mapping f may require additional parameters Z, M, rational_map, E', and/or iso_map. These MUST be specified when applicable.

All applications MUST choose a domain separation tag (DST) in accordance with the guidelines in Section 3.1. In addition, applications whose security requires a random oracle that returns points on the target curve MUST use a suite whose encoding type is hash_to_curve (Section 3); see Section 8.9.

The below table lists the curves for which suites are defined and the subsection that gives the corresponding parameters.

E	Section
NIST P-256	Section 8.1
NIST P-384	Section 8.2
NIST P-521	Section 8.3
curve25519 / edwards25519	Section 8.4
curve448 / edwards448	Section 8.5
secp256k1	Section 8.6
BLS12-381	Section 8.7

Table 2

8.1. Suites for NIST P-256

This section defines ciphersuites for the NIST P-256 elliptic curve [FIPS186-4].

P256_XMD:SHA-256_SSWU_RO_ is defined as follows:

*encoding type: hash_to_curve (Section 3)

*E: y^2 = x^3 + A * x + B, where

-A = -3

-B = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b

*p: 2^256 - 2^224 + 2^192 + 2^96 - 1

*m: 1

*k: 128

*sgn0: sgn0_le (Section 4.1.2)

*expand_message: expand_message_xmd (Section 5.3.1)

*H: SHA-256

*L: 48

*f: Simplified SWU method, Section 6.6.2

*Z: -10

*h_eff: 1

P256_XMD:SHA-256_SVDW_RO_ is identical to P256_XMD:SHA-256_SSWU_RO_, except for the following parameters:

*f: Shallue-van de Woestijne method, Section 6.6.1

*Z: -3

P256_XMD:SHA-256_SSWU_NU_ is identical to P256_XMD:SHA-256_SSWU_RO_, except that the encoding type is encode_to_curve (Section 3).

P256_XMD:SHA-256_SVDW_NU_ is identical to P256_XMD:SHA-256_SVDW_RO_, except that the encoding type is encode_to_curve (Section 3).

An optimized example implementation of the Simplified SWU mapping to P-256 is given in Appendix D.2.

8.2. Suites for NIST P-384

This section defines ciphersuites for the NIST P-384 elliptic curve [FIPS186-4].

P384_XMD:SHA-512_SSWU_RO_ is defined as follows:

*encoding type: hash_to_curve (Section 3)

*E: y^2 = x^3 + A * x + B, where

-A = -3

-B = 0xb3312fa7e23ee7e4988e056be3f82d19181d9c6efe8141120314088f5013875ac656398d8a2ed19d2a

*p: 2^384 - 2^128 - 2^96 + 2^32 - 1

*m: 1

*k: 192

*sgn0: sgn0_le (Section 4.1.2)

*expand_message: expand_message_xmd (Section 5.3.1)

*H: SHA-512

*L: 72

*f: Simplified SWU method, Section 6.6.2

*Z: -12

*h_eff: 1

P384_XMD:SHA-512_SVDW_RO_ is identical to P384_XMD:SHA-512_SSWU_RO_, except for the following parameters:

*f: Shallue-van de Woestijne method, Section 6.6.1

*Z: -1

P384_XMD:SHA-512_SSWU_NU_ is identical to P384_XMD:SHA-512_SSWU_RO_, except that the encoding type is encode_to_curve (Section 3).

P384_XMD:SHA-512_SVDW_NU_ is identical to P384_XMD:SHA-512_SVDW_RO_, except that the encoding type is encode_to_curve (Section 3).

An optimized example implementation of the Simplified SWU mapping to P-384 is given in Appendix D.2.

8.3. Suites for NIST P-521

This section defines ciphersuites for the NIST P-521 elliptic curve [FIPS186-4].

P521_XMD:SHA-512_SSWU_RO_ is defined as follows:

*encoding type: hash_to_curve (Section 3)

*E: y^2 = x^3 + A * x + B, where

-A = -3

-B =
0x51953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef109e156193951ec7e937b

*p: 2^521 - 1

*m: 1

*k: 256

*sgn0: sgn0_le (Section 4.1.2)

*expand_message: expand_message_xmd (Section 5.3.1)

*H: SHA-512

*L: 96

*f: Simplified SWU method, Section 6.6.2

*Z: -4

*h_eff: 1

P521_XMD:SHA-512_SVDW_RO_ is identical to P521_XMD:SHA-512_SSWU_RO_, except for the following parameters:

*f: Shallue-van de Woestijne method, Section 6.6.1

*Z: 1

P521_XMD:SHA-512_SSWU_NU_ is identical to P512_XMD:SHA-512_SSWU_RO_, except that the encoding type is encode_to_curve (Section 3).

P521_XMD:SHA-512_SVDW_NU_ is identical to P512_XMD:SHA-512_SVDW_RO_, except that the encoding type is encode_to_curve (Section 3).

An optimized example implementation of the Simplified SWU mapping to P-521 is given in Appendix D.2.

8.4. Suites for curve25519 and edwards25519

This section defines ciphersuites for curve25519 and edwards25519 [RFC7748].

curve25519_XMD:SHA-256_ELL2_RO_ is defined as follows:

*encoding type: hash_to_curve (Section 3)

  *E: K * t^2 = s^3 + J * s^2 + s, where

    -J = 486662

    -K = 1

  *p: 2^255 - 19

  *m: 1

  *k: 128

  *sgn0: sgn0_le (Section 4.1.2)

  *expand_message: expand_message_xmd (Section 5.3.1)

  *H: SHA-256

  *L: 48

  *f: Elligator 2 method, Section 6.7.1

  *Z: 2

  *h_eff: 8

edwards25519_XMD:SHA-256_ELL2_RO_ is identical to
curve25519_XMD:SHA-256_ELL2_RO_, except for the following
parameters:

  *E: a * v^2 + w^2 = 1 + d * v^2 * w^2, where

    -a = -1

    -d =
     0x52036cee2b6ffe738cc740797779e89800700a4d4141d8ab75eb4dca135978a3

  *f: Twisted Edwards Elligator 2 method, Section 6.8.2

  *M: curve25519 defined in [RFC7748], Section 4.1

  *rational_map: the birational map defined in [RFC7748], Section
   4.1

curve25519_XMD:SHA-256_ELL2_NU_ is identical to
curve25519_XMD:SHA-256_ELL2_RO_, except that the encoding type is
encode_to_curve (Section 3).

edwards25519_XMD:SHA-256_ELL2_NU_ is identical to edwards25519_XMD:SHA-256_ELL2_RO_, except that ene encoding type is encode_to_curve (Section 3).

curve25519_XMD:SHA-512_ELL2_RO_ is identical to curve25519_XMD:SHA-256_ELL2_RO_, except that H is SHA-512.

curve25519_XMD:SHA-512_ELL2_NU_ is identical to curve25519_XMD:SHA-256_ELL2_NU_, except that H is SHA-512.

edwards25519_XMD:SHA-512_ELL2_RO_ is identical to edwards25519_XMD:SHA-256_ELL2_RO_, except that H is SHA-512.

edwards25519_XMD:SHA-512_ELL2_NU_ is identical to edwards25519_XMD:SHA-256_ELL2_NU_, except that H is SHA-512.

Optimized example implementations of the above mappings are given in Appendix D.3 and Appendix D.4.

8.5. Suites for curve448 and edwards448

This section defines ciphersuites for curve448 and edwards448 [RFC7748].

curve448_XMD:SHA-512_ELL2_RO_ is defined as follows:

- encoding type: hash_to_curve (Section 3)
- E: K * t^2 = s^3 + J * s^2 + s, where
  - -J = 156326
  - -K = 1
- p: 2^448 - 2^224 - 1
- m: 1
- k: 224
- sgn0: sgn0_le (Section 4.1.2)
- expand_message: expand_message_xmd (Section 5.3.1)
- H: SHA-512
- L: 84
- f: Elligator 2 method, Section 6.7.1
- Z: -1

  *h_eff: 4

edwards448_XMD:SHA-512_ELL2_RO_ is identical to curve448_XMD:SHA-512_ELL2_RO_, except for the following parameters:

  *E: a * v^2 + w^2 = 1 + d * v^2 * w^2, where

    -a = 1

    -d = -39081

  *f: Twisted Edwards Elligator 2 method, Section 6.8.2

  *M: curve448, defined in [RFC7748], Section 4.2

  *rational_map: the 4-isogeny map defined in [RFC7748], Section 4.2

curve448_XMD:SHA-512_ELL2_NU_ is identical to curve448_XMD:SHA-512_ELL2_RO_, except that the encoding type is encode_to_curve (Section 3).

edwards448_XMD:SHA-512_ELL2_NU_ is identical to edwards448_XMD:SHA-512_ELL2_RO_, except that the encoding type is encode_to_curve (Section 3).

Optimized example implementations of the above mappings are given in Appendix D.5 and Appendix D.6.

8.6. Suites for secp256k1

This section defines ciphersuites for the secp256k1 elliptic curve [SEC2].

secp256k1_XMD:SHA-256_SSWU_RO_ is defined as follows:

  *encoding type: hash_to_curve (Section 3)

  *E: y^2 = x^3 + 7

  *p: 2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1

  *m: 1

  *k: 128

  *sgn0: sgn0_le (Section 4.1.2)

  *expand_message: expand_message_xmd (Section 5.3.1)

  *H: SHA-256

*L: 48

*f: Simplified SWU for AB == 0, Section 6.6.3

*Z: -11

*E': y'^2 = x'^3 + A' * x' + B', where

-A':
0x3f8731abdd661adca08a5558f0f5d272e953d363cb6f0e5d405447c01a444533

-B': 1771

*iso_map: the 3-isogeny map from E' to E given in Appendix C.1

*h_eff: 1

secp256k1_XMD:SHA-256_SVDW_RO_ is identical to secp256k1_XMD:SHA-256_SSWU_RO_, except for the following parameters:

*f: Shallue-van de Woestijne method, Section 6.6.1

*Z: 1

*E' is not required for this suite

*iso_map is not required for this suite

secp256k1_XMD:SHA-256_SSWU_NU_ is identical to secp256k1_XMD:SHA-256_SSWU_RO_, except that the encoding type is encode_to_curve (Section 3).

secp256k1_XMD:SHA-256_SVDW_NU_ is identical to secp256k1_XMD:SHA-256_SVDW_RO_, except that the encoding type is encode_to_curve (Section 3).

An optimized example implementation of the Simplified SWU mapping to the curve E' isogenous to secp256k1 is given in Appendix D.2.

8.7. Suites for BLS12-381

This section defines ciphersuites for groups G1 and G2 of the BLS12-381 elliptic curve [BLS12-381].

8.7.1. BLS12-381 G1

BLS12381G1_XMD:SHA-256_SSWU_RO_ is defined as follows:

*encoding type: hash_to_curve (Section 3)

  *E: y^2 = x^3 + 4

  *p:
   0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fe

  *m: 1

  *k: 128

  *sgn0: sgn0_be (Section 4.1.1)

  *expand_message: expand_message_xmd (Section 5.3.1)

  *H: SHA-256

  *L: 64

  *f: Simplified SWU for AB == 0, Section 6.6.3

  *Z: 11

  *E': y'^2 = x'^3 + A' * x' + B', where

     -A' =
      0x144698a3b8e9433d693a02c96d4982b0ea985383ee66a8d8e8981aefd881ac98936f8da0e0f97f5cf4

     -B' =
      0x12e2908d11688030018b12e8753eee3b2016c1f0f24f4070a0b9c14fcef35ef55a23215a316ceaa5d1

  *iso_map: the 11-isogeny map from E' to E given in Appendix C.2

  *h_eff: 0xd201000000010001

BLS12381G1_XMD:SHA-256_SVDW_RO_ is identical to
BLS12381G1_XMD:SHA-256_SSWU_RO_, except for the following
parameters:

  *f: Shallue-van de Woestijne method, Section 6.6.1

  *Z: -3

  *E' is not required for this suite

  *iso_map is not required for this suite

BLS12381G1_XMD:SHA-256_SSWU_NU_ is identical to
BLS12381G1_XMD:SHA-256_SSWU_RO_, except that the encoding type is
encode_to_curve (Section 3).

BLS12381G1_XMD:SHA-256_SVDW_NU_ is identical to BLS12381G1_XMD:SHA-256_SVDW_RO_, except that the encoding type is encode_to_curve (Section 3).

Note that the h_eff values for these suites are chosen for compatibility with the fast cofactor clearing method described by Scott ([WB19] Section 5).

An optimized example implementation of the Simplified SWU mapping to the curve E' isogenous to BLS12-381 G1 is given in Appendix D.2.

8.7.2. BLS12-381 G2

BLS12381G2_XMD:SHA-256_SSWU_RO_ is defined as follows:

- encoding type: hash_to_curve (Section 3)
- E: y^2 = x^3 + 4 * (1 + I)
- base field F is GF(p^m), where
  - p: 0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9
  - m: 2
  - (1, I) is the basis for F, where I^2 + 1 == 0 in F
- k: 128
- sgn0: sgn0_be (Section 4.1.1)
- expand_message: expand_message_xmd (Section 5.3.1)
- H: SHA-256
- L: 64
- f: Simplified SWU for AB == 0, Section 6.6.3
- Z: -(2 + I)
- E': y'^2 = x'^3 + A' * x' + B', where
  - A' = 240 * I
  - B' = 1012 * (1 + I)
- iso_map: the isogeny map from E' to E given in Appendix C.3

*h_eff:
0xbc69f08f2ee75b3584c6a0ea91b352888e2a8e9145ad7689986ff031508ffe1329c2f178731db956d82bf

BLS12381G2_XMD:SHA-256_SVDW_RO_ is identical to BLS12381G2_XMD:SHA-256_SSWU_RO_, except for the following parameters:

*f: Shallue-van de Woestijne method, Section 6.6.1

*Z: I

*E' is not required for this suite

*iso_map is not required for this suite

BLS12381G2_XMD:SHA-256_SSWU_NU_ is identical to BLS12381G2_XMD:SHA-256_SSWU_RO_, except that the encoding type is encode_to_curve (Section 3).

BLS12381G2_XMD:SHA-256_SVDW_NU_ is identical to BLS12381G2_XMD:SHA-256_SVDW_RO_, except that the encoding type is encode_to_curve (Section 3).

Note that the h_eff values for these suites are chosen for compatibility with the fast cofactor clearing method described by Budroni and Pintore ([BP17], Section 4.1).

8.8. Defining a new hash-to-curve suite

The RECOMMENDED way to define a new hash-to-curve suite is:

1. E, F, p, and m are determined by the elliptic curve and its base field; k is determined by the security level of the elliptic curve.

2. Choose encoding type, either hash_to_curve or encode_to_curve (Section 3).

3. Choose a sgn0 variant following the guidelines in Section 4.1.

4. Compute L as described in Section 5.1.

5. Choose an expand_message variant from Section 5.3 plus any underlying cryptographic primitives (e.g., a hash function H).

6. Choose a mapping following the guidelines in Section 6.1, and select any required parameters for that mapping.

7. Choose h_eff to be either the cofactor of E or, if a fast cofactor clearing method is to be used, a value appropriate to that method as discussed in Section 7.

8. Construct a Suite ID following the guidelines in Section 8.9.

When hashing to an elliptic curve not listed in this section, corresponding hash-to-curve suites SHOULD be fully specified as described above.

8.9. Suite ID naming conventions

Suite IDs MUST be constructed as follows:

```
CURVE_ID || "_" || HASH_ID || "_" || MAP_ID || "_" || ENC_VAR || "_"
```

The fields CURVE_ID, HASH_ID, MAP_ID, and ENC_VAR are ASCII-encoded strings of at most 64 characters each. Fields MUST contain only ASCII characters between 0x21 and 0x7E (inclusive) other underscore (i.e., 0x5f).

As indicated above, each field (including the last) is followed by an underscore ("_", ASCII 0x5f). This helps to ensure that Suite IDs are prefix free. Suite IDs MUST include the final underscore and MUST NOT include any characters after the final underscore.

Suite ID fields MUST be chosen as follows:

* CURVE_ID: a human-readable representation of the target elliptic curve.

* HASH_ID: a human-readable representation of the expand_message function and any underlying hash primitives used in hash_to_field (Section 5). This field MUST be constructed as follows:

  ```
  EXP_TAG || ":" || HASH_NAME
  ```

  EXP_TAG indicates the expand_message variant:

  - "XMD" for expand_message_xmd (Section 5.3.1).

  - "XOF" for expand_message_xof (Section 5.3.2).

  HASH_NAME is a human-readable name for the underlying hash primitive. As examples:

  1. For expand_message_xof (Section 5.3.2) with SHAKE-128, HASH_ID is "XOF:SHAKE-128".

2. For expand_message_xmd (Section 5.3.1) with SHA3-256, HASH_ID is "XMD:SHA3-256".

*MAP_ID: a human-readable representation of the map_to_curve function as defined in Section 6. These are defined as follows:

-"SVDW" for or Shallue and van de Woestijne (Section 6.6.1).

-"SSWU" for Simplified SWU (Section 6.6.2, Section 6.6.3).

-"ELL2" for Elligator 2 (Section 6.7.1, Section 6.8.2).

*ENC_VAR: a string indicating the encoding type and other information. The first two characters of this string indicate whether the suite represents a hash_to_curve or an encode_to_curve operation (Section 3), as follows:

-If ENC_VAR begins with "RO", the suite uses hash_to_curve.

-If ENC_VAR begins with "NU", the suite uses encode_to_curve.

-ENC_VAR MUST NOT begin with any other string.

ENC_VAR MAY also be used to encode other information used to identify variants, for example, a version number. The RECOMMENDED way to do so is to add one or more subfields separated by colons. For example, "RO:V02" is an appropriate ENC_VAR value for the second version of a random-oracle suite, while "RO:V02:FOO01:BAR17" might be used to indicate a variant of that suite.

9. IANA Considerations

This document has no IANA actions.

10. Security Considerations

When constant-time implementations are required, all basic operations and utility functions must be implemented in constant time, as discussed in Section 4. In some applications (e.g., embedded systems), leakage through other side channels (e.g., power or electromagnetic side channels) may be pertinent. Defending against such leakage is outside the scope of this document, because the nature of the leakage and the appropriate defense depends on the protocol from which a hash-to-curve function is invoked.

Section 3.1 describes considerations related to domain separation.

Section 5 describes considerations for uniformly hashing to field elements; see Section 10.1 and Section 10.2 for further discussion.

Each encoding variant (Section 3) accepts an arbitrary byte string and maps it to a pseudorandom point on the curve. Note, however, that directly evaluating the mappings of Section 6 produces an output that is distinguishable from random.

When the hash_to_curve function (Section 3) is instantiated with a hash_to_field function that is indifferentiable from a random oracle (Section 5), the resulting function is indifferentiable from a random oracle ([FFSTV13], [LBB19], [MRH04]). In most cases such a function can be safely used in protocols whose security analysis assumes a random oracle that outputs points on an elliptic curve. As Ristenpart et al. discuss in [RSS11], however, not all security proofs that rely on random oracles continue to hold when those oracles are replaced by indifferentiable functionalities. This limitation should be considered when analyzing the security of protocols relying on the hash_to_curve function.

When hashing passwords using any function described in this document, an adversary who learns the output of the hash function (or potentially any intermediate value, e.g., the output of hash_to_field) may be able to carry out a dictionary attack. To mitigate such attacks, it is recommended to first execute a more costly key derivation function (e.g., PBKDF2 [RFC2898] or scrypt [RFC7914]) on the password, then hash the output of that function to the target elliptic curve. For collision resistance, the hash underlying the key derivation function should be chosen according to the guidelines listed in Section 5.3.

10.1. hash_to_field security

The hash_to_field function defined in Section 5 is indifferentiable from a random oracle [MRH04] when expand_message (Section 5.3) is modeled as a random oracle. By composability of indifferentiability proofs, this also holds when expand_message is proved indifferentiable from a random oracle relative to an underlying primitive that is modeled as a random oracle. When following the guidelines in Section 5.3, both variants of expand_message defined in that section meet this requirement (see also Section 10.2).

We very briefly sketch the indifferentiability argument for hash_to_field. Notice that each integer mod p that hash_to_field returns (i.e., each element of the vector representation of F) is a member of an equivalence class of roughly 2^k integers of length log2(p) + k bits, all of which are equal modulo p. For each integer mod p that hash_to_field returns, the simulator samples one member of this equivalence class at random and outputs the byte string returned by I2OSP. (Notice that this is essentially the inverse of the hash_to_field procedure.)

Finally, the expand_message variants in this document (Section 5.3) always append the domain separation tag DST to the strings hashed by H, the underlying hash or extensible output function. This means that invocations of H outside of hash_to_field can be separated from those inside of hash_to_field by appending a tag distinct from DST to their inputs. Other expand_message variants that follow the guidelines in Section 5.3.3 are expected to have similar properties, but these should be analyzed on a case-by-case basis.

10.2. expand_message_xmd security

The expand_message_xmd function defined in Section 5.3.1 is indifferentiable from a random oracle [MRH04] when one of the following holds:

1. H is indifferentiable from a random oracle,

2. H is a sponge-based hash function whose inner function is modeled as a random transformation or random permutation [BDPV08], or

3. H is a Merkle-Damgaard hash function whose compression function is modeled as a random oracle [CDMP05].

For cases (1) and (2), the indifferentiability of expand_message_xmd follows directly from the indifferentiability of H.

For case (3), i.e., for H a Merkle-Damgaard hash function, indifferentiability follows from [CDMP05], Theorem 3.5. In particular, expand_message_xmd computes b_0 by prepending one block of 0-bytes to the message and auxiliary information (length, counter, and DST). Then, each of the output blocks b_i, i >= 1 in expand_message_xmd is the result of invoking H on a unique, prefix-free encoding of b_0. This is true, first, because the length of the input to all such invocations is equal and fixed by the choice of H and DST, and second, because each such input has a unique suffix (because of the inclusion of the counter byte I2OSP(i, 1)).

The essential difference between the construction of [CDMP05] and expand_message_xmd is that the latter hashes a counter appended to strxor(b_0, b_(i - 1)) (step 9) rather than to b_0. This approach increases the Hamming distance between inputs to different invocations of H, which reduces the likelihood that nonidealities in H affect the distribution of the b_i values.

11. Acknowledgements

The authors would like to thank Adam Langley for his detailed writeup of Elligator 2 with Curve25519 [L13]; Dan Boneh, Christopher Patton, and Benjamin Lipp for educational discussions; and Sean

Devlin, Justin Drake, Dan Harkins, Thomas Icart, Andy Polyakov, Leonid Reyzin, Michael Scott, and Mathy Vanhoef for helpful feedback.

12. Contributors

*Sharon Goldberg

Boston University

goldbe@cs.bu.edu

*Ela Lee

Royal Holloway, University of London

Ela.Lee.2010@live.rhul.ac.uk

13. References

13.1. 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>.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <https://www.rfc-editor.org/info/rfc2898>.

[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>.

[RFC7914] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", RFC 7914, DOI 10.17487/RFC7914, August 2016, <https://www.rfc-editor.org/info/rfc7914>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

13.2. Informative References

[AFQTZ14] Aranha, D.F., Fouque, P.A., Qian, C., Tibouchi, M., and J.C. Zapalowicz, "Binary Elligator squared", DOI 10.1007/978-3-319-13051-4_2, pages 20-37, In Selected Areas in Cryptography - SAC 2014, 2014, <https://doi.org/10.1007/978-3-319-13051-4_2>.

[AR13] Adj, G. and F. Rodriguez-Henriquez, "Square Root Computation over Even Extension Fields", DOI 10.1109/TC.2013.145, pages 2829-2841, In IEEE Transactions on Computers. vol 63 issue 11, November 2014, <https://doi.org/10.1109/TC.2013.145>.

[BBJLP08] Bernstein, D.J., Birkner, P., Joye, M., Lange, T., and C. Peters, "Twisted Edwards curves", DOI 10.1007/978-3-540-68164-9_26, pages 389-405, In AFRICACRYPT 2008, 2008, <https://doi.org/10.1007/978-3-540-68164-9_26>.

[BCIMRT10] Brier, E., Coron, J-S., Icart, T., Madore, D., Randriam, H., and M. Tibouchi, "Efficient Indifferentiable Hashing into Ordinary Elliptic Curves", DOI 10.1007/978-3-642-14623-7_13, pages 237-254, In Advances in Cryptology - CRYPTO 2010, 2010, <https://doi.org/10.1007/978-3-642-14623-7_13>.

[BDPV08] Bertoni,, G., Daemen, J., Peeters, M., and G. Van Assche, "On the Indifferentiability of the Sponge Construction", DOI 10.1007/978-3-540-78967-3_11, pages 181-197, In Advances in Cryptology - EUROCRYPT 2008, 2008, <https://doi.org/10.1007/978-3-540-78967-3_11>.

[BF01] Boneh, D. and M. Franklin, "Identity-based encryption from the Weil pairing", DOI 10.1007/3-540-44647-8_13, pages 213-229, In Advances in Cryptology - CRYPTO 2001, August 2001, <https://doi.org/10.1007/3-540-44647-8_13>.

[BHKL13] Bernstein, D.J., Hamburg, M., Krasnova, A., and T. Lange, "Elligator: elliptic-curve points indistinguishable from uniform random strings", DOI 10.1145/2508859.2516734, pages 967-980, In Proceedings of the 2013 ACM SIGSAC conference on computer and communications security., November 2013, <https://doi.org/10.1145/2508859.2516734>.

[BLAKE2X] Aumasson, J-P., Neves, S., Wilcox-O'Hearn, Z., and C. Winnerlein, "BLAKE2X", December 2016, <https://blake2.net/blake2x.pdf>.

[BLMP19] Bernstein, D.J., Lange, T., Martindale, C., and L. Panny, "Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies", DOI 10.1007/978-3-030-17656-3, In Advances in Cryptology - EUROCRYPT 2019, 2019, <https://doi.org/10.1007/978-3-030-17656-3>.

[BLS01] Boneh, D., Lynn, B., and H. Shacham, "Short signatures from the Weil pairing", DOI 10.1007/s00145-004-0314-9, pages 297-319, In Journal of Cryptology, vol 17, July 2004, <https://doi.org/10.1007/s00145-004-0314-9>.

[BLS03] Barreto, P., Lynn, B., and M. Scott, "Constructing Elliptic Curves with Prescribed Embedding Degrees", DOI 10.1007/3-540-36413-7_19, pages 257-267, In Security in Communication Networks, 2003, <https://doi.org/10.1007/3-540-36413-7_19>.

[BLS12-381] Bowe, S., "BLS12-381: New zk-SNARK Elliptic Curve Construction", March 2017, <https://electriccoin.co/blog/new-snark-curve/>.

[BMP00] Boyko, V., MacKenzie, P.D., and S. Patel, "Provably secure password-authenticated key exchange using Diffie-Hellman", DOI 10.1007/3-540-45539-6_12, pages 156-171, In Advances in Cryptology - EUROCRYPT 2000, May 2000, <https://doi.org/10.1007/3-540-45539-6_12>.

[BN05] Barreto, P. and M. Naehrig, "Pairing-Friendly Elliptic Curves of Prime Order", DOI 10.1007/11693383_22, pages 319-331, In Selected Areas in Cryptography 2005, 2006, <https://doi.org/10.1007/11693383_22>.

[BP17] Budroni, A. and F. Pintore, "Efficient hash maps to G2 on BLS curves", ePrint 2017/419, May 2017, <https://eprint.iacr.org/2017/419>.

[C93] Cohen, H., "A Course in Computational Algebraic Number Theory", ISBN 9783642081422, publisher Springer-Verlag, 1993, <https://doi.org/10.1007/978-3-662-02945-9>.

[CDMP05] Coron, J-S., Dodis, Y., Malinaud, C., and P. Puniya, "Merkle-Damgaard Revisited: How to Construct a Hash Function", DOI 10.1007/11535218_26, pages 430-448, In Advances in Cryptology - CRYPTO 2005, 2005, <https://doi.org/10.1007/11535218_26>.

[CFADLNV05] Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., and F. Vercauteren, "Handbook of Elliptic and Hyperelliptic Curve Cryptography", ISBN 9781584885184,

publisher Chapman and Hall / CRC, 2005, <https://www.crcpress.com/9781584885184>.

[CK11] Couveignes, J. and J. Kammerer, "The geometry of flex tangents to a cubic curve and its parameterizations", DOI 10.1016/j.jsc.2011.11.003, pages 266-281, In Journal of Symbolic Computation, vol 47 issue 3, 2012, <https://doi.org/10.1016/j.jsc.2011.11.003>.

[CN08] Chang, D. and M. Nandi, "Improved indifferentiability security analysis for ChopMD hash function", DOI 10.1007/978-3-540-71039-4_27, pages 429-443, In FSE, 2008, <https://doi.org/10.1007/978-3-540-71039-4_27>.

[DFL12] Daubignard, M., Fouque, P-A., and Y. Lakhnech, "Generic indifferentiability proofs of hash designs", DOI 10.1109/CSF.2012.13, pages 340-353, In CSF, 2012, <https://doi.org/10.1109/CSF.2012.13>.

[F11] Farashahi, R.R., "Hashing into Hessian curves", DOI 10.1007/978-3-642-21969-6_17, pages 278-289, In AFRICACRYPT 2011, 2011, <https://doi.org/10.1007/978-3-642-21969-6_17>.

[FFSTV13] Farashahi, R.R., Fouque, P.A., Shparlinski, I.E., Tibouch, M., and J.F. Voloch, "Indifferentiable deterministic hashing to elliptic and hyperelliptic curves", DOI 10.1090/S0025-5718-2012-02606-8, pages 491-512, In Math. Comp. vol 82, 2013, <https://doi.org/10.1090/S0025-5718-2012-02606-8>.

[FIPS180-4] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[FIPS186-4] National Institute of Standards and Technology (NIST), "FIPS Publication 186-4: Digital Signature Standard", July 2013, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[FIPS202] National Institute of Standards and Technology (NIST), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", August 2015, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.

[FJT13] Fouque, P-A., Joux, A., and M. Tibouchi, "Injective encodings to elliptic curves", DOI 10.1007/978-3-642-39059-3_14, pages 203-218, In ACISP

2013, 2013, <https://doi.org/10.1007/978-3-642-39059-3_14>.

[FKR11] Fuentes-Castaneda, L., Knapp, E., and F. Rodriguez-Henriquez, "Fast Hashing to G2 on Pairing-Friendly Curves", DOI 10.1007/978-3-642-28496-0_25, pages 412-430, In Selected Areas in Cryptography, 2011, <https://doi.org/10.1007/978-3-642-28496-0_25>.

[FSV09] Farashahi, R.R., Shparlinski, I.E., and J.F. Voloch, "On hashing into elliptic curves", DOI 10.1515/JMC.2009.022, pages 353-360, In Journal of Mathematical Cryptology, vol 3 no 4, 2009, <https://doi.org/10.1515/JMC.2009.022>.

[FT10] Fouque, P-A. and M. Tibouchi, "Estimating the size of the image of deterministic hash functions to elliptic curves.", DOI 10.1007/978-3-642-14712-8_5, pages 81-91, In Progress in Cryptology - LATINCRYPT 2010, 2010, <https://doi.org/10.1007/978-3-642-14712-8_5>.

[FT12] Fouque, P-A. and M. Tibouchi, "Indifferentiable Hashing to Barreto-Naehrig Curves", DOI 10.1007/978-3-642-33481-8_1, pages 1-7, In Progress in Cryptology - LATINCRYPT 2012, 2012, <https://doi.org/10.1007/978-3-642-33481-8_1>.

[hash2curve-repo] "Hashing to Elliptic Curves - GitHub repository", 2019, <https://github.com/cfrg/draft-irtf-cfrg-hash-to-curve>.

[Icart09] Icart, T., "How to Hash into Elliptic Curves", DOI 10.1007/978-3-642-03356-8_18, pages 303-316, In Advances in Cryptology - CRYPTO 2009, 2009, <https://doi.org/10.1007/978-3-642-03356-8_18>.

[J96] Jablon, D.P., "Strong password-only authenticated key exchange", DOI 10.1145/242896.242897, pages 5-26, In SIGCOMM Computer Communication Review, vol 26 issue 5, 1996, <https://doi.org/10.1145/242896.242897>.

[jubjub-fq] "zkcrypto/jubjub - fq.rs", 2019, <https://github.com/zkcrypto/jubjub/blob/master/src/fq.rs>.

[KLR10] Kammerer, J., Lercier, R., and G. Renault, "Encoding points on hyperelliptic curves over finite fields in deterministic polynomial time", DOI 10.1007/978-3-642-17455-1_18, pages 278-297, In PAIRING 2010, 2010, <https://doi.org/10.1007/978-3-642-17455-1_18>.

[L13] Langley, A., "Implementing Elligator for Curve25519", 2013, <https://www.imperialviolet.org/2013/12/25/elligator.html>.

[LBB19] Lipp, B., Blanchet, B., and K. Bhargavan, "A Mechanised Proof of the WireGuard Virtual Private Network Protocol", In INRIA Research Report No. 9269, April 2019, <https://hal.inria.fr/hal-02100345/>.

[MRH04] Maurer, U., Renner, R., and C. Holenstein, "Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology", DOI 10.1007/978-3-540-24638-1_2, pages 21-39, In TCC 2004: Theory of Cryptography, February 2004, <https://doi.org/10.1007/978-3-540-24638-1_2>.

[MT07] Maurer, U. and S. Tessaro, "Domain extension of public random functions: Beyond the birthday barrier", DOI 10.1007/978-3-540-74143-5_11, pages 187-204, In Advances in Cryptology - CRYPTO 2007, 2007, <https://doi.org/10.1007/978-3-540-74143-5_11>.

[p1363a] IEEE Computer Society, "IEEE Standard Specifications for Public-Key Cryptography---Amendment 1: Additional Techniques", March 2004, <https://standards.ieee.org/standard/1363a-2004.html>.

[RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <https://www.rfc-editor.org/info/rfc7693>.

[RSS11] Ristenpart, T., Shacham, H., and T. Shrimpton, "Careful with Composition: Limitations of the Indifferentiability Framework", DOI 10.1007/978-3-642-20465-4_27, pages 487-506, In Advances in Cryptology - EUROCRYPT 2011, May 2011, <https://doi.org/10.1007/978-3-642-20465-4_27>.

[S05] Skalba, M., "Points on elliptic curves over finite fields", DOI 10.4064/aa117-3-7, pages 293-301, In Acta Arithmetica, vol 117 no 3, 2005, <https://doi.org/10.4064/aa117-3-7>.

[S85] Schoof, R., "Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p", DOI 10.1090/S0025-5718-1985-0777280-6, pages 483-494, In Mathematics of Computation vol 44 issue 170, April 1985, <https://doi.org/10.1090/S0025-5718-1985-0777280-6>.

[SAGE] The Sage Developers, "SageMath, the Sage Mathematics Software System", 2019, <https://www.sagemath.org>.

[SBCDK09] Scott, M., Benger, N., Charlemagne, M., Dominguez Perez, L.J., and E.J. Kachisa, "Fast Hashing to G2 on Pairing-Friendly Curves", DOI 10.1007/978-3-642-03298-1_8, pages 102-113, In Pairing-Based Cryptography - Pairing 2009, 2009, <https://doi.org/10.1007/978-3-642-03298-1_8>.

[SEC1] Standards for Efficient Cryptography Group (SECG), "SEC 1: Elliptic Curve Cryptography", May 2009, <http://www.secg.org/sec1-v2.pdf>.

[SEC2] Standards for Efficient Cryptography Group (SECG), "SEC 2: Recommended Elliptic Curve Domain Parameters", January 2010, <http://www.secg.org/sec2-v2.pdf>.

[SS04] Schinzel, A. and M. Skalba, "On equations y^2 = x^n + k in a finite field.", DOI 10.4064/ba52-3-1, pages 223-226, In Bulletin Polish Acad. Sci. Math. vol 52, no 3, 2004, <https://doi.org/10.4064/ba52-3-1>.

[SW06] Shallue, A. and C. van de Woestijne, "Construction of rational points on elliptic curves over finite fields", DOI 10.1007/11792086_36, pages 510-524, In Algorithmic Number Theory. ANTS 2006., 2006, <https://doi.org/10.1007/11792086_36>.

[T14] Tibouchi, M., "Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings", DOI 10.1007/978-3-662-45472-5_10, pages 139-156, In Financial Cryptography and Data Security - FC 2014, 2014, <https://doi.org/10.1007/978-3-662-45472-5_10>.

[TK17] Tibouchi, M. and T. Kim, "Improved elliptic curve hashing and point representation", DOI 10.1007/s10623-016-0288-2, pages 161-177, In Designs, Codes, and Cryptography, vol 82, 2017, <https://doi.org/10.1007/s10623-016-0288-2>.

[U07] Ulas, M., "Rational points on certain hyperelliptic curves over finite fields", DOI 10.4064/ba55-2-1, pages 97-104, In Bulletin Polish Acad. Sci. Math. vol 55, no 2, 2007, <https://doi.org/10.4064/ba55-2-1>.

[W08] Washington, L.C., "Elliptic curves: Number theory and cryptography", ISBN 9781420071467, publisher Chapman and

Hall / CRC, edition 2nd, 2008, <https://www.crcpress.com/9781420071467>.

[W19] Wahby, R.S., "An explicit, generic parameterization for the Shallue--van de Woestijne map", 2019, <https://github.com/cfrg/draft-irtf-cfrg-hash-to-curve/doc/svdw_params.pdf>.

[WB19] Wahby, R.S. and D. Boneh, "Fast and simple constant-time hashing to the BLS12-381 elliptic curve", DOI 10.13154/tches.v2019.i4.154-179, ePrint 2019/403, issue 4, volume 2019, In IACR Trans. CHES, August 2019, <https://eprint.iacr.org/2019/403>.

[x9.62] ANSI, "Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62-1998, September 1998.

Appendix A. Related Work

The problem of mapping arbitrary bit strings to elliptic curve points has been the subject of both practical and theoretical research. This section briefly describes the background and research results that underly the recommendations in this document. This section is provided for informational purposes only.

A naive but generally insecure method of mapping a string msg to a point on an elliptic curve E having n points is to first fix a point P that generates the elliptic curve group, and a hash function Hn from bit strings to integers less than n; then compute Hn(msg) * P, where the * operator represents scalar multiplication. The reason this approach is insecure is that the resulting point has a known discrete log relationship to P. Thus, except in cases where this method is specified by the protocol, it must not be used; doing so risks catastrophic security failures.

Boneh et al. [BLS01] describe an encoding method they call MapToGroup, which works roughly as follows: first, use the input string to initialize a pseudorandom number generator, then use the generator to produce a pseudorandom value x in F. If x is the x-coordinate of a point on the elliptic curve, output that point. Otherwise, generate a new pseudorandom value x in F and try again. Since a random value x in F has probability about 1/2 of corresponding to a point on the curve, the expected number of tries is just two. However, the running time of this method depends on the input string, which means that it is not safe to use in protocols sensitive to timing side channels.

Schinzel and Skalba [SS04] introduce a method of constructing elliptic curve points deterministically, for a restricted class of curves and a very small number of points. Skalba [S05] generalizes this construction to more curves and more points on those curves. Shallue and van de Woestijne [SW06] further generalize and simplify Skalba's construction, yielding concretely efficient maps to a constant fraction of the points on almost any curve. Fouque and Tibouchi [FT12] give a parameterization of this mapping for Barreto-Naehrig pairing-friendly curves [BN05].

Ulas [U07] describes a simpler version of the Shallue-van de Woestijne map, and Brier et al. [BCIMRT10] give a further simplification, which the authors call the "simplified SWU" map. That simplified map applies only to fields of characteristic p = 3 (mod 4); Wahby and Boneh [WB19] generalize to fields of any characteristic, and give further optimizations.

Boneh and Franklin give a deterministic algorithm mapping to certain supersingular curves over fields of characteristic p = 2 (mod 3) [BF01]. Icart gives another deterministic algorithm which maps to any curve over a field of characteristic p = 2 (mod 3) [Icart09]. Several extensions and generalizations follow this work, including [FSV09], [FT10], [KLR10], [F11], and [CK11].

Following the work of Farashahi [F11], Fouque et al. [FJT13] describe a mapping to curves over fields of characteristic p = 3 (mod 4) having a number of points divisible by 4. Bernstein et al. [BHKL13] optimize this mapping and describe a related mapping that they call "Elligator 2," which applies to any curve over a field of odd characteristic having a point of order 2. This includes Curve25519 and Curve448, both of which are CFRG-recommended curves [RFC7748]. Bernstein et al. [BLMP19] extend the Elligator 2 map to a class of supersingular curves over fields of characteristic p = 3 (mod 4).

An important caveat regarding all of the above deterministic mapping functions is that none of them map to the entire curve, but rather to some fraction of the points. This means that they cannot be used directly to construct a random oracle that outputs points on the curve.

Brier et al. [BCIMRT10] give two solutions to this problem. The first, which Brier et al. prove applies to Icart's method, computes f(H0(msg)) + f(H1(msg)) for two distinct hash functions H0 and H1 from bit strings to F and a mapping f from F to the elliptic curve E. The second, which applies to essentially all deterministic mappings but is more costly, computes f(H0(msg)) + H2(msg) * P, for P a generator of the elliptic curve group and H2 a hash from bit strings to integers modulo r, the order of the elliptic curve group.

Farashahi et al. [FFSTV13] improve the analysis of the first method, showing that it applies to essentially all deterministic mappings. Tibouchi and Kim [TK17] further refine the analysis and describe additional optimizations.

Complementary to the problem of mapping from bit strings to elliptic curve points, Bernstein et al. [BHKL13] study the problem of mapping from elliptic curve points to uniformly random bit strings, giving solutions for a class of curves including Montgomery and twisted Edwards curves. Tibouchi [T14] and Aranha et al. [AFQTZ14] generalize these results. This document does not deal with this complementary problem.

Appendix B. Rational maps

This section gives several useful rational maps.

B.1. Twisted Edwards to Montgomery curves

This section gives a generic birational map between twisted Edwards and Montgomery curves. This birational map comprises the rational map specified in Section 6.8.1 and its inverse.

The twisted Edwards curve

```
a * v^2 + w^2 = 1 + d * v^2 * w^2
```

is birationally equivalent to the Montgomery curve

```
K * t^2 = s^3 + J * s^2 + s
```

by the following mappings ([BBJLP08], Theorem 3.2). To convert from twisted Edwards to Montgomery form, the mapping is

```
*J = 2 * (a + d) / (a - d)

*K = 4 / (a - d)

*s = (1 + w) / (1 - w)

*t = (1 + w) / (v * (1 - w))
```

This mapping is defined when a != d, which is guaranteed by the definition of twisted Edwards curves. The mapping is undefined when v == 0 or w == 1. If (v, w) == (0, -1), return the point (s, t) = (0, 0). For all other undefined inputs, return the identity point on the Montgomery curve. (This follows from [BBJLP08], Section 3.)

To convert from Montgomery to twisted Edwards form, the mapping is

*a = (J + 2) / K

*d = (J - 2) / K

*v = s / t

*w = (s - 1) / (s + 1)

This mapping is defined when J != 2, J != -2, and K != 0; all Montgomery curves meet these criteria. The mapping is undefined when t == 0 or s == -1. If (s, t) == (0, 0), return the point (v, w) = (0, -1). For all other undefined inputs, return the identity point on the twisted Edwards curve, namely, (v, w) = (0, 1). (This follows from [BBJLP08], Section 3.)

(Note that Section 6.8.1 gives a simpler rule for handling undefined inputs to this rational map: always return the identity point. The simpler rule gives the same result when used as part of an encoding function (Section 3), because the cofactor clearing step will always map the point (v, w) = (0, -1) to the identity point.)

Composing the mapping of this section with the mapping from Montgomery to Weierstrass curves in Appendix B.2 yields a mapping from twisted Edwards curves to Weierstrass curves, which is the form required by the mappings in Section 6.6. This composition of mappings can be used to apply the Shallue-van de Woestijne (Section 6.6.1) or Simplified SWU (Section 6.6.2) method to twisted Edwards curves.

B.2. Montgomery to Weierstrass curves

The rational map from the point (s, t) on the Montgomery curve

K * t^2 = s^3 + J * s^2 + s

to the point (x, y) on the equivalent Weierstrass curve

y^2 = x^3 + A * x + B

is given by:

*A = (3 - J^2) / (3 * K^2)

*B = (2 * J^3 - 9 * J) / (27 * K^3)

*x = (3 * s + J) / (3 * K)

*y = t / K

The inverse map, from the point (x, y) to the point (s, t), is given by

*s = (3 * K * x - J) / 3

*t = y * K

This mapping can be used to apply the Shallue-van de Woestijne (Section 6.6.1) or Simplified SWU (Section 6.6.2) method to Montgomery curves.

Appendix C. Isogeny maps for Suites

This section specifies the isogeny maps for the secp256k1 and BLS12-381 suites listed in Section 8.

These maps are given in terms of affine coordinates. Wahby and Boneh ([WB19], Section 4.3) show how to evaluate these maps in a projective coordinate system (Appendix D.1), which avoids modular inversions.

Refer to the draft repository [hash2curve-repo] for a Sage [SAGE] script that constructs these isogenies.

C.1. 3-isogeny map for secp256k1

This section specifies the isogeny map for the secp256k1 suite listed in Section 8.6.

The 3-isogeny map from (x', y') on E' to (x, y) on E is given by the following rational functions:

*x = x_num / x_den, where

-x_num = k_(1,3) * x'^3 + k_(1,2) * x'^2 + k_(1,1) * x' + k_(1,0)

-x_den = x'^2 + k_(2,1) * x' + k_(2,0)

*y = y' * y_num / y_den, where

-y_num = k_(3,3) * x'^3 + k_(3,2) * x'^2 + k_(3,1) * x' + k_(3,0)

-y_den = x'^3 + k_(4,2) * x'^2 + k_(4,1) * x' + k_(4,0)

The constants used to compute x_num are as follows:

*k_(1,0) =
0x8e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38daaaaa8c7

*k_(1,1) =
0x7d3d4c80bc321d5b9f315cea7fd44c5d595d2fc0bf63b92dfff1044f17c6581

*k_(1,2) =
0x534c328d23f234e6e2a413deca25caece4506144037c40314ecbd0b53d9dd262

*k_(1,3) =
0x8e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38daaaaa88c

The constants used to compute x_den are as follows:

*k_(2,0) =
0xd35771193d94918a9ca34ccbb7b640dd86cd409542f8487d9fe6b745781eb49b

*k_(2,1) =
0xedadc6f64383dc1df7c4b2d51b54225406d36b641f5e41bbc52a56612a8c6d14

The constants used to compute y_num are as follows:

*k_(3,0) =
0x4bda12f684bda12f684bda12f684bda12f684bda12f684bda12f684b8e38e23c

*k_(3,1) =
0xc75e0c32d5cb7c0fa9d0a54b12a0a6d5647ab046d686da6fdffc90fc201d71a3

*k_(3,2) =
0x29a6194691f91a73715209ef6512e576722830a201be2018a765e85a9ecee931

*k_(3,3) =
0x2f684bda12f684bda12f684bda12f684bda12f684bda12f684bda12f38e38d84

The constants used to compute y_den are as follows:

*k_(4,0) =
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffeffffff93b

*k_(4,1) =
0x7a06534bb8bdb49fd5e9e6632722c2989467c1bfc8e8d978dfb425d2685c2573

*k_(4,2) =
0x6484aa716545ca2cf3a70c3fa8fe337e0a3d21162f0d6299a7bf8192bfd2a76f

C.2. 11-isogeny map for BLS12-381 G1

The 11-isogeny map from (x', y') on E' to (x, y) on E is given by the following rational functions:

*x = x_num / x_den, where

-x_num = k_(1,11) * x'^11 + k_(1,10) * x'^10 + k_(1,9) * x'^9 + ... + k_(1,0)

-x_den = x'^10 + k_(2,9) * x'^9 + k_(2,8) * x'^8 + ... + k_(2,0)

*y = y' * y_num / y_den, where

-y_num = k_(3,15) * x'^15 + k_(3,14) * x'^14 + k_(3,13) * x'^13 + ... + k_(3,0)

-y_den = x'^15 + k_(4,14) * x'^14 + k_(4,13) * x'^13 + ... + k_(4,0)

The constants used to compute x_num are as follows:

*k_(1,0) =
0x11a05f2b1e833340b809101dd99815856b303e88a2d7005ff2627b56cdb4e2c85610c2d5f2e62d6eaeac1

*k_(1,1) =
0x17294ed3e943ab2f0588bab22147a81c7c17e75b2f6a8417f565e33c70d1e86b4838f2a6f318c356e834e

*k_(1,2) =
0xd54005db97678ec1d1048c5d10a9a1bce032473295983e56878e501ec68e25c958c3e3d2a09729fe0179f

*k_(1,3) =
0x1778e7166fcc6db74e0609d307e55412d7f5e4656a8dbf25f1b33289f1b330835336e25ce3107193c5b38

*k_(1,4) =
0xe99726a3199f4436642b4b3e4118e5499db995a1257fb3f086eeb65982fac18985a286f301e77c451154c

*k_(1,5) =
0x1630c3250d7313ff01d1201bf7a74ab5db3cb17dd952799b9ed3ab9097e68f90a0870d2dcae73d19cd13c

*k_(1,6) =
0xd6ed6553fe44d296a3726c38ae652bfb11586264f0f8ce19008e218f9c86b2a8da25128c1052ecaddd7f2

*k_(1,7) =
0x17b81e7701abdbe2e8743884d1117e53356de5ab275b4db1a682c62ef0f2753339b7c8f8c8f475af9ccb5

*k_(1,8) =
0x80d3cf1f9a78fc47b90b33563be990dc43b756ce79f5574a2c596c928c5d1de4fa295f296b74e956d7198

*k_(1,9) =
0x169b1f8e1bcfa7c42e0c37515d138f22dd2ecb803a0c5c99676314baf4bb1b7fa3190b2edc0327797f241

*k_(1,10) =
0x10321da079ce07e272d8ec09d2565b0dfa7dccdde6787f96d50af36003b14866f69b771f8c285decca67d

*k_(1,11) =
0x6e08c248e260e70bd1e962381edee3d31d79d7e22c837bc23c0bf1bc24c6b68c24b1b80b64d391fa9c8ba

The constants used to compute x_den are as follows:

*k_(2,0) =
0x8ca8d548cff19ae18b2e62f4bd3fa6f01d5ef4ba35b48ba9c9588617fc8ac62b558d681be343df8993cf9

*k_(2,1) =
0x12561a5deb559c4348b4711298e536367041e8ca0cf0800c0126c2588c48bf5713daa8846cb026e9e5c82

*k_(2,2) =
0xb2962fe57a3225e8137e629bff2991f6f89416f5a718cd1fca64e00b11aceacd6a3d0967c94fedcfcc239

*k_(2,3) =
0x3425581a58ae2fec83aafef7c40eb545b08243f16b1655154cca8abc28d6fd04976d5243eecf5c4130de8

*k_(2,4) =
0x13a8e162022914a80a6f1d5f43e7a07dffdfc759a12062bb8d6b44e833b306da9bd29ba81f35781d539d3

*k_(2,5) =
0xe7355f8e4e667b955390f7f0506c6e9395735e9ce9cad4d0a43bcef24b8982f7400d24bc4228f11c02df9

*k_(2,6) =
0x772caacf16936190f3e0c63e0596721570f5799af53a1894e2e073062aede9cea73b3538f0de06cec2574

*k_(2,7) =
0x14a7ac2a9d64a8b230b3f5b074cf01996e7f63c21bca68a81996e1cdf9822c580fa5b9489d11e2d311f7c

*k_(2,8) =
0xa10ecf6ada54f825e920b3dafc7a3cce07f8d1d7161366b74100da67f39883503826692abba43704776ec

*k_(2,9) =
0x95fc13ab9e92ad4476d6e3eb3a56680f682b4ee96f7d03776df533978f31c1593174e4b4b7865002d6384

The constants used to compute y_num are as follows:

*k_(3,0) =
0x90d97c81ba24ee0259d1f094980dcfa11ad138e48a869522b52af6c956543d3cd0c7aee9b3ba3c2be9845

*k_(3,1) =
0x134996a104ee5811d51036d776fb46831223e96c254f383d0f906343eb67ad34d6c56711962fa8bfe097e

*k_(3,2) =
0xcc786baa966e66f4a384c86a3b49942552e2d658a31ce2c344be4b91400da7d26d521628b00523b8dfe24

*k_(3,3) =
0x1f86376e8981c217898751ad8746757d42aa7b90eeb791c09e4a3ec03251cf9de405aba9ec61deca6355c

*k_(3,4) =
0x8cc03fdefe0ff135caf4fe2a21529c4195536fbe3ce50b879833fd221351adc2ee7f8dc099040a841b6da

*k_(3,5) =
0x16603fca40634b6a2211e11db8f0a6a074a7d0d4afadb7bd76505c3d3ad5544e203f6326c95a807299b23

*k_(3,6) =
0x4ab0b9bcfac1bbcb2c977d027796b3ce75bb8ca2be184cb5231413c4d634f3747a87ac2460f415ec961f8

*k_(3,7) =
0x987c8d5333ab86fde9926bd2ca6c674170a05bfe3bdd81ffd038da6c26c842642f64550fedfe935a15e4c

*k_(3,8) =
0x9fc4018bd96684be88c9e221e4da1bb8f3abd16679dc26c1e8b6e6a1f20cabe69d65201c78607a360370e

*k_(3,9) =
0xe1bba7a1186bdb5223abde7ada14a23c42a0ca7915af6fe06985e7ed1e4d43b9b3f7055dd4eba6f2bafaa

*k_(3,10) =
0x19713e47937cd1be0dfd0b8f1d43fb93cd2fcbcb6caf493fd1183e416389e61031bf3a5cce3fbafce8137

*k_(3,11) =
0x18b46a908f36f6deb918c143fed2edcc523559b8aaf0c2462e6bfe7f911f643249d9cdf41b44d606ce07c

*k_(3,12) =
0xb182cac101b9399d155096004f53f447aa7b12a3426b08ec02710e807b4633f06c851c1919211f20d4c04

*k_(3,13) =
0x245a394ad1eca9b72fc00ae7be315dc757b3b080d4c158013e6632d3c40659cc6cf90ad1c232a6442d9d3

*k_(3,14) =
0x5c129645e44cf1102a159f748c4a3fc5e673d81d7e86568d9ab0f5d396a7ce46ba1049b6579afb7866b1e

*k_(3,15) =
0x15e6be4e990f03ce4ea50b3b42df2eb5cb181d8f84965a3957add4fa95af01b2b665027efec01c7704b45

The constants used to compute y_den are as follows:

*k_(4,0) =
0x16112c4c3a9c98b252181140fad0eae9601a6de578980be6eec3232b5be72e7a07f3688ef60c206d01479

*k_(4,1) =
0x1962d75c2381201e1a0cbd6c43c348b885c84ff731c4d59ca4a10356f453e01f78a4260763529e3532f61

*k_(4,2) =
0x58df3306640da276faaae7d6e8eb15778c4855551ae7f310c35a5dd279cd2eca6757cd636f96f891e2538

*k_(4,3) =
0x16b7d288798e5395f20d23bf89edb4d1d115c5dbddbcd30e123da489e726af41727364f2c28297ada8d26

*k_(4,4) =
0xbe0e079545f43e4b00cc912f8228ddcc6d19c9f0f69bbb0542eda0fc9dec916a20b15dc0fd2ededda3914

*k_(4,5) =
0x8d9e5297186db2d9fb266eaac783182b70152c65550d881c5ecd87b6f0f5a6449f38db9dfa9cce202c647

*k_(4,6) =
0x166007c08a99db2fc3ba8734ace9824b5eecfdfa8d0cf8ef5dd365bc400a0051d5fa9c01a58b1fb93d1a1

*k_(4,7) =
0x16a3ef08be3ea7ea03bcddfabba6ff6ee5a4375efa1f4fd7feb34fd206357132b920f5b00801dee460ee4

*k_(4,8) =
0x1866c8ed336c61231a1be54fd1d74cc4f9fb0ce4c6af5920abc5750c4bf39b4852cfe2f7bb9248836b233

*k_(4,9) =
0x167a55cda70a6e1cea820597d94a84903216f763e13d87bb5308592e7ea7d4fbc7385ea3d529b35e346ef

*k_(4,10) =
0x4d2f259eea405bd48f010a01ad2911d9c6dd039bb61a6290e591b36e636a5c871a5c29f4f83060400f8b4

*k_(4,11) =
0xaccbb67481d033ff5852c1e48c50c477f94ff8aefce42d28c0f9a88cea7913516f968986f7ebbea9684b5

*k_(4,12) =
0xad6b9514c767fe3c3613144b45f1496543346d98adf02267d5ceef9a00d9b8693000763e3b90ac11e99b1

*k_(4,13) =
0x2660400eb2e4f3b628bdd0d53cd76f2bf565b94e72927c1cb748df27942480e420517bd8714cc80d1fadc

*k_(4,14) =
0xe0fa1d816ddc03e6b24255e0d7819c171c40f65e273b853324efcd6356caa205ca2f570f13497804415 47

C.3. 3-isogeny map for BLS12-381 G2

The 3-isogeny map from (x', y') on E' to (x, y) on E is given by the following rational functions:

*x = x_num / x_den, where

-x_num = k_(1,3) * x'^3 + k_(1,2) * x'^2 + k_(1,1) * x' + k_(1,0)

-x_den = x'^2 + k_(2,1) * x' + k_(2,0)

*y = y' * y_num / y_den, where

-y_num = k_(3,3) * x'^3 + k_(3,2) * x'^2 + k_(3,1) * x' + k_(3,0)

-y_den = x'^3 + k_(4,2) * x'^2 + k_(4,1) * x' + k_(4,0)

The constants used to compute x_num are as follows:

*k_(1,0) =
0x5c759507e8e333ebb5b7a9a47d7ed8532c52d39fd3a042a88b58423c50ae15d5c2638e343d9c71c6238a
+
0x5c759507e8e333ebb5b7a9a47d7ed8532c52d39fd3a042a88b58423c50ae15d5c2638e343d9c71c6238a
* I

*k_(1,1) =
0x11560bf17baa99bc32126fced787c88f984f87adf7ae0c7f9a208c6b4f20a4181472aaa9cb8d555526a9
* I

*k_(1,2) =
0x11560bf17baa99bc32126fced787c88f984f87adf7ae0c7f9a208c6b4f20a4181472aaa9cb8d555526a9
+
0x8ab05f8bdd54cde190937e76bc3e447cc27c3d6fbd7063fcd104635a790520c0a395554e5c6aaaa9354f
* I

*k_(1,3) =
0x171d6541fa38ccfaed6dea691f5fb614cb14b4e7f4e810aa22d6108f142b85757098e38d0f671c7188e2

The constants used to compute x_den are as follows:

*k_(2,0) =
0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fe
* I

*k_(2,1) = 0xc +
0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fe
* I

The constants used to compute y_num are as follows:

```
*k_(3,0) =
 0x1530477c7ab4113b59a4c18b076d11930f7da5d4a07f649bf54439d87d27e500fc8c25ebf8c92f6812cfc
 +
 0x1530477c7ab4113b59a4c18b076d11930f7da5d4a07f649bf54439d87d27e500fc8c25ebf8c92f6812cfc
 * I

*k_(3,1) =
 0x5c759507e8e333ebb5b7a9a47d7ed8532c52d39fd3a042a88b58423c50ae15d5c2638e343d9c71c6238aa
 * I

*k_(3,2) =
 0x11560bf17baa99bc32126fced787c88f984f87adf7ae0c7f9a208c6b4f20a4181472aaa9cb8d555526a9f
 +
 0x8ab05f8bdd54cde190937e76bc3e447cc27c3d6fbd7063fcd104635a790520c0a395554e5c6aaaa9354f
 * I

*k_(3,3) =
 0x124c9ad43b6cf79bfbf7043de3811ad0761b0f37a1e26286b0e977c69aa274524e79097a56dc4bd9e1b37
```

The constants used to compute y_den are as follows:

```
*k_(4,0) =
 0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fef
 +
 0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fef
 * I

*k_(4,1) =
 0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fef
 * I

*k_(4,2) = 0x12 +
 0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9fef
 * I
```

Appendix D. Sample Code

This section gives sample implementations optimized for some of the elliptic curves listed in Section 8. Sample Sage [SAGE] code for each algorithm can also be found in the draft repository [hash2curve-repo].

D.1. Interface and projective coordinate systems

The sample code in this section uses a different interface than the mappings of Section 6. Specifically, each mapping function in this section has the following signature:

```
(xn, xd, yn, yd) = map_to_curve(u)
```

The resulting point (x, y) is given by (xn / xd, yn / yd).

The reason for this modified interface is that it enables further optimizations when working with points in a projective coordinate system. This is desirable, for example, when the resulting point will be immediately multiplied by a scalar, since most scalar multiplication algorithms operate on projective points.

The following are two commonly used projective coordinate systems and the corresponding conversions:

- A point (X, Y, Z) in homogeneous projective coordinates corresponds to the affine point (x, y) = (X / Z, Y / Z); the inverse conversion is given by (X, Y, Z) = (x, y, 1). To convert (xn, xd, yn, yd) to homogeneous projective coordinates, compute (X, Y, Z) = (xn * yd, yn * xd, xd * yd).

- A point (X', Y', Z') in Jacobian projective coordinates corresponds to the affine point (x, y) = (X' / Z'^2, Y' / Z'^3); the inverse conversion is given by (X', Y', Z') = (x, y, 1). To convert (xn, xd, yn, yd) to Jacobian projective coordinates, compute (X', Y', Z') = (xn * xd * yd^2, yn * yd^2 * xd^3, xd * yd).

D.2. Simplified SWU for p = 3 (mod 4)

The following is a straight-line implementation of the Simplified SWU mapping that applies to any curve over GF(p) for p = 3 (mod 4). This includes the ciphersuites for NIST curves P-256, P-384, and P-521 [FIPS186-4] given in Section 8. It also includes the curves isogenous to secp256k1 (Section 8.6) and BLS12-381 G1 (Section 8.7.1).

The implementations for these curves differ only in the constants and the base field. The constant definitions below are given in terms of the parameters for the Simplified SWU mapping; for parameter values for the curves listed above, see Section 8.1 (P-256), Section 8.2 (P-384), Section 8.3 (P-521), Section 8.6 (E' isogenous to secp256k1), and Section 8.7.1 (E' isogenous to BLS12-381 G1).

```
map_to_curve_simple_swu_3mod4(u)

Input: u, an element of F.
Output: (xn, xd, yn, yd) such that (xn / xd, yn / yd) is a
        point on the target curve.

Constants: defined per curve; see above.
1.  c1 = (p - 3) / 4           # Integer arithmetic
2.  c2 = sqrt(-Z^3)

Steps:
1.  tv1 = u^2
2.  tv3 = Z * tv1
3.  tv2 = tv3^2
4.   xd = tv2 + tv3
5.  x1n = xd + 1
6.  x1n = x1n * B
7.   xd = -A * xd
8.   e1 = xd == 0
9.   xd = CMOV(xd, Z * A, e1)  # If xd == 0, set xd = Z * A
10. tv2 = xd^2
11. gxd = tv2 * xd             # gxd == xd^3
12. tv2 = A * tv2
13. gx1 = x1n^2
14. gx1 = gx1 + tv2            # x1n^2 + A * xd^2
15. gx1 = gx1 * x1n            # x1n^3 + A * x1n * xd^2
16. tv2 = B * gxd
17. gx1 = gx1 + tv2            # x1n^3 + A * x1n * xd^2 + B * xd^3
18. tv4 = gxd^2
19. tv2 = gx1 * gxd
20. tv4 = tv4 * tv2            # gx1 * gxd^3
21.  y1 = tv4^c1               # (gx1 * gxd^3)^((p - 3) / 4)
22.  y1 = y1 * tv2             # gx1 * gxd * (gx1 * gxd^3)^((p - 3) / 4)
23. x2n = tv3 * x1n            # x2 = x2n / xd = Z * u^2 * x1n / xd
24.  y2 = y1 * c2              # y2 = y1 * sqrt(-Z^3)
25.  y2 = y2 * tv1
26.  y2 = y2 * u
27. tv2 = y1^2
28. tv2 = tv2 * gxd
29.  e2 = tv2 == gx1
30.  xn = CMOV(x2n, x1n, e2)   # If e2, x = x1, else x = x2
31.   y = CMOV(y2, y1, e2)     # If e2, y = y1, else y = y2
32.  e3 = sgn0(u) == sgn0(y)   # Fix sign of y
33.   y = CMOV(-y, y, e3)
34. return (xn, xd, y, 1)
```

D.3. curve25519 (Elligator 2)

The following is a straight-line implementation of Elligator 2 for curve25519 [RFC7748] as specified in Section 8.4.

```
map_to_curve_elligator2_curve25519(u)

Input: u, an element of F.
Output: (xn, xd, yn, yd) such that (xn / xd, yn / yd) is a
        point on curve25519.

Constants:
1. c1 = (p + 3) / 8           # Integer arithmetic
2. c2 = 2^c1
3. c3 = sqrt(-1)
4. c4 = (p - 5) / 8           # Integer arithmetic

Steps:
1.  tv1 = u^2
2.  tv1 = 2 * tv1
3.   xd = tv1 + 1             # Nonzero: -1 is square (mod p), tv1 is no
4.  x1n = -486662             # x1 = x1n / xd = -486662 / (1 + 2 * u^2)
5.  tv2 = xd^2
6.  gxd = tv2 * xd            # gxd = xd^3
7.  gx1 = 486662 * xd         # 486662 * xd
8.  gx1 = gx1 + x1n           # x1n + 486662 * xd
9.  gx1 = gx1 * x1n           # x1n^2 + 486662 * x1n * xd
10. gx1 = gx1 + tv2           # x1n^2 + 486662 * x1n * xd + xd^2
11. gx1 = gx1 * x1n           # x1n^3 + 486662 * x1n^2 * xd + x1n * xd^2
12. tv3 = gxd^2
13. tv2 = tv3^2               # gxd^4
14. tv3 = tv3 * gxd           # gxd^3
15. tv3 = tv3 * gx1           # gx1 * gxd^3
16. tv2 = tv2 * tv3           # gx1 * gxd^7
17. y11 = tv2^c4              # (gx1 * gxd^7)^((p - 5) / 8)
18. y11 = y11 * tv3           # gx1 * gxd^3 * (gx1 * gxd^7)^((p - 5) / 8
19. y12 = y11 * c3
20. tv2 = y11^2
21. tv2 = tv2 * gxd
22.  e1 = tv2 == gx1
23.  y1 = CMOV(y12, y11, e1)  # If g(x1) is square, this is its sqrt
24. x2n = x1n * tv1           # x2 = x2n / xd = 2 * u^2 * x1n / xd
25. y21 = y11 * u
26. y21 = y21 * c2
27. y22 = y21 * c3
28. gx2 = gx1 * tv1           # g(x2) = gx2 / gxd = 2 * u^2 * g(x1)
29. tv2 = y21^2
30. tv2 = tv2 * gxd
31.  e2 = tv2 == gx2
32.  y2 = CMOV(y22, y21, e2)  # If g(x2) is square, this is its sqrt
33. tv2 = y1^2
34. tv2 = tv2 * gxd
35.  e3 = tv2 == gx1
36.  xn = CMOV(x2n, x1n, e3)  # If e3, x = x1, else x = x2
```

```
  y = CMOV(y2, y1, e3)    # If e3, y = y1, else y = y2
 e4 = sgn0(u) == sgn0(y)  # Fix sign of y
  y = CMOV(-y, y, e4)
return (xn, xd, y, 1)
```

D.4. edwards25519 (Elligator 2)

The following is a straight-line implementation of Elligator 2 for edwards25519 [RFC7748] as specified in Section 8.4. The subroutine map_to_curve_elligator2_curve25519 is defined in Appendix D.3.

```
map_to_curve_elligator2_edwards25519(u)

Input: u, an element of F.
Output: (xn, xd, yn, yd) such that (xn / xd, yn / yd) is a
        point on edwards25519.

Constants:
1. c1 = sqrt(-486664)    # sgn0(c1) MUST equal 1

Steps:
1.  (xMn, xMd, yMn, yMd) = map_to_curve_elligator2_curve25519(u)
2.  xn = xMn * yMd
3.  xn = xn * c1
4.  xd = xMd * yMn       # xn / xd = c1 * xM / yM
5.  yn = xMn - xMd
6.  yd = xMn + xMd       # (n / d - 1) / (n / d + 1) = (n - d) / (n + d)
7. tv1 = xd * yd
8.   e = tv1 == 0
9.  xn = CMOV(xn, 0, e)
10. xd = CMOV(xd, 1, e)
11. yn = CMOV(yn, 1, e)
12. yd = CMOV(yd, 1, e)
13. return (xn, xd, yn, yd)
```

D.5. curve448 (Elligator 2)

The following is a straight-line implementation of Elligator 2 for curve448 [RFC7748] as specified in Section 8.5.

```
map_to_curve_elligator2_curve448(u)

Input: u, an element of F.
Output: (xn, xd, yn, yd) such that (xn / xd, yn / yd) is a
        point on curve448.

Constants:
1. c1 = (p - 3) / 4           # Integer arithmetic

Steps:
1.  tv1 = u^2
2.   e1 = tv1 == 1
3.  tv1 = CMOV(tv1, 0, e1)    # If Z * u^2 == -1, set tv1 = 0
4.   xd = 1 - tv1
5.  x1n = -156326
6.  tv2 = xd^2
7.  gxd = tv2 * xd            # gxd = xd^3
8.  gx1 = 156326 * xd         # 156326 * xd
9.  gx1 = gx1 + x1n           # x1n + 156326 * xd
10. gx1 = gx1 * x1n           # x1n^2 + 156326 * x1n * xd
11. gx1 = gx1 + tv2           # x1n^2 + 156326 * x1n * xd + xd^2
12. gx1 = gx1 * x1n           # x1n^3 + 156326 * x1n^2 * xd + x1n * xd^2
13. tv3 = gxd^2
14. tv2 = gx1 * gxd           # gx1 * gxd
15. tv3 = tv3 * tv2           # gx1 * gxd^3
16.  y1 = tv3^c1              # (gx1 * gxd^3)^((p - 3) / 4)
17.  y1 = y1 * tv2            # gx1 * gxd * (gx1 * gxd^3)^((p - 3) / 4)
18. x2n = -tv1 * x1n          # x2 = x2n / xd = -1 * u^2 * x1n / xd
19.  y2 = y1 * u
20.  y2 = CMOV(y2, 0, e1)
21. tv2 = y1^2
22. tv2 = tv2 * gxd
23.  e2 = tv2 == gx1
24.  xn = CMOV(x2n, x1n, e2)  # If e2, x = x1, else x = x2
25.   y = CMOV(y2, y1, e2)    # If e2, y = y1, else y = y2
26.  e3 = sgn0(u) == sgn0(y)  # Fix sign of y
27.   y = CMOV(-y, y, e3)
28. return (xn, xd, y, 1)
```

D.6. edwards448 (Elligator 2)

The following is a straight-line implementation of Elligator 2 for edwards448 [RFC7748] as specified in Section 8.5. The subroutine map_to_curve_elligator2_curve448 is defined in Appendix D.5.

```
map_to_curve_elligator2_edwards448(u)

Input: u, an element of F.
Output: (xn, xd, yn, yd) such that (xn / xd, yn / yd) is a
        point on edwards448.

Steps:
1. (xn, xd, yn, yd) = map_to_curve_elligator2_curve448(u)
2.  xn2 = xn^2
3.  xd2 = xd^2
4.  xd4 = xd2^2
5.  yn2 = yn^2
6.  yd2 = yd^2
7.  xEn = xn2 - xd2
8.  tv2 = xEn - xd2
9.  xEn = xEn * xd2
10. xEn = xEn * yd
11. xEn = xEn * yn
12. xEn = xEn * 4
13. tv2 = tv2 * xn2
14. tv2 = tv2 * yd2
15. tv3 = 4 * yn2
16. tv1 = tv3 + yd2
17. tv1 = tv1 * xd4
18. xEd = tv1 + tv2
19. tv2 = tv2 * xn
20. tv4 = xn * xd4
21. yEn = tv3 - yd2
22. yEn = yEn * tv4
23. yEn = yEn - tv2
24. tv1 = xn2 + xd2
25. tv1 = tv1 * xd2
26. tv1 = tv1 * xd
27. tv1 = tv1 * yn2
28. tv1 = -2 * tv1
29. yEd = tv2 + tv1
30. tv4 = tv4 * yd2
31. yEd = yEd + tv4
32. tv1 = xEd * yEd
33.   e = tv1 == 0
34. xEn = CMOV(xEn, 0, e)
35. xEd = CMOV(xEd, 1, e)
36. yEn = CMOV(yEn, 1, e)
37. yEd = CMOV(yEd, 1, e)
38. return (xEn, xEd, yEn, yEd)
```

Appendix E. Scripts for parameter generation

This section gives Sage [SAGE] scripts used to generate parameters for the mappings of Section 6.

E.1. Finding Z for the Shallue and van de Woestijne map

The below function outputs an appropriate Z for the Shallue and van de Woestijne map (Section 6.6.1).

```
# Arguments:
# - F, a field object, e.g., F = GF(2^521 - 1)
# - A and B, the coefficients of the curve equation y^2 = x^3 + A * x +
def find_z_svdw(F, A, B):
    g = lambda x: F(x)^3 + F(A) * F(x) + F(B)
    h = lambda Z: -(F(3) * Z^2 + F(4) * A) / (F(4) * g(Z))
    ctr = F.gen()
    while True:
        for Z_cand in (F(ctr), F(-ctr)):
            if g(Z_cand) == F(0):
                # Criterion 1: g(Z) != 0 in F.
                continue
            if h(Z_cand) == F(0):
                # Criterion 2: -(3 * Z^2 + 4 * A) / (4 * g(Z)) != 0 in F
                continue
            if not h(Z_cand).is_square():
                # Criterion 3: -(3 * Z^2 + 4 * A) / (4 * g(Z)) is square
                continue
            if g(Z_cand).is_square() or g(-Z_cand / F(2)).is_square():
                # Criterion 4: At least one of g(Z) and g(-Z / 2) is squ
                return Z_cand
        ctr += 1
```

E.2. Finding Z for Simplified SWU

The below function outputs an appropriate Z for the Simplified SWU map (Section 6.6.2).

```
# Arguments:
# - F, a field object, e.g., F = GF(2^521 - 1)
# - A and B, the coefficients of the curve equation y^2 = x^3 + A * x +
def find_z_sswu(F, A, B):
    R.<xx> = F[]                        # Polynomial ring over F
    g = xx^3 + F(A) * xx + F(B)         # y^2 = g(x) = x^3 + A * x + B
    ctr = F.gen()
    while True:
        for Z_cand in (F(ctr), F(-ctr)):
            if Z_cand.is_square():
                # Criterion 1: Z is non-square in F.
                continue
            if Z_cand == F(-1):
                # Criterion 2: Z != -1 in F.
                continue
            if not (g - Z_cand).is_irreducible():
                # Criterion 3: g(x) - Z is irreducible over F.
                continue
            if g(B / (Z_cand * A)).is_square():
                # Criterion 4: g(B / (Z * A)) is square in F.
                return Z_cand
        ctr += 1
```

E.3. Finding Z for Elligator 2

The below function outputs an appropriate Z for the Elligator 2 map (Section 6.7.1).

```
# Argument:
# - F, a field object, e.g., F = GF(2^255 - 19)
def find_z_ell2(F):
    ctr = F.gen()
    while True:
        for Z_cand in (F(ctr), F(-ctr)):
            if Z_cand.is_square():
                # Z must be a non-square in F.
                continue
            return Z_cand
        ctr += 1
```

Appendix F. sqrt functions

This section defines special-purpose sqrt functions for the three most common cases, q = 3 (mod 4), q = 5 (mod 8), and q = 9 (mod 16). In addition, it gives a generic constant-time algorithm that works for any prime modulus.

[AR13] and [S85] describe optimized methods for extension fields.

F.1. q = 3 (mod 4)

```
sqrt_3mod4(x)

Parameters:
- F, a finite field of characteristic p and order q = p^m.

Input: x, an element of F.
Output: z, an element of F such that (z^2) == x, if x is square in F.

Constants:
1. c1 = (q + 1) / 4     # Integer arithmetic

Procedure:
1. return x^c1
```

F.2. q = 5 (mod 8)

```
sqrt_5mod8(x)

Parameters:
- F, a finite field of characteristic p and order q = p^m.

Input: x, an element of F.
Output: z, an element of F such that (z^2) == x, if x is square in F.

Constants:
1. c1 = sqrt(-1) in F, i.e., (c1^2) == -1 in F
2. c2 = (q + 3) / 8     # Integer arithmetic

Procedure:
1. tv1 = x^c2
2. tv2 = tv1 * c1
3.   e = (tv1^2) == x
4.   z = CMOV(tv2, tv1, e)
5. return z
```

F.3. q = 9 (mod 16)

```
sqrt_9mod16(x)

Parameters:
- F, a finite field of characteristic p and order q = p^m.

Input: x, an element of F.
Output: z, an element of F such that (z^2) == x, if x is square in F.

Constants:
1. c1 = sqrt(-1) in F, i.e., (c1^2) == -1 in F
2. c2 = sqrt(c1) in F, i.e., (c2^2) == c1 in F
3. c3 = sqrt(-c1) in F, i.e., (c3^2) == -c1 in F
4. c4 = (q + 7) / 16         # Integer arithmetic

Procedure:
1. tv1 = x^c4
2. tv2 = c1 * tv1
3. tv3 = c2 * tv1
4. tv4 = c3 * tv1
5.  e1 = (tv2^2) == x
6.  e2 = (tv3^2) == x
7. tv1 = CMOV(tv1, tv2, e1)  # Select tv2 if (tv2^2) == x
8. tv2 = CMOV(tv4, tv3, e2)  # Select tv3 if (tv3^2) == x
9.  e3 = (tv2^2) == x
10.  z = CMOV(tv1, tv2, e3)  # Select the sqrt from tv1 and tv2
11. return z
```

F.4. Constant-time Tonelli-Shanks algorithm

This algorithm is a constant-time version of the classic Tonelli-Shanks algorithm ([C93], Algorithm 1.5.1) due to Sean Bowe, Jack Grigg, and Eirik Ogilvie-Wigley [jubjub-fq], adapted and optimized by Michael Scott.

This algorithm applies to GF(p) for any p. Note, however, that the special-purpose algorithms given in the prior sections are faster, when they apply.

```
sqrt_ts_ct(x)

Parameters:
- F, a finite field of characteristic p and order q = p^m.

Input x, an element of F.
Output: z, an element of F such that z^2 == x, if x is square in F.

Constants:
1. c1, the largest integer such that 2^c1 divides q - 1.
2. c2 = (q - 1) / (2^c1)        # Integer arithmetic
3. c3 = (c2 - 1) / 2            # Integer arithmetic
4. c4, a non-square value in F
5. c5 = c4^c2 in F

Procedure:
1.  z = x^c3
2.  t = z * z * x
3.  z = z * x
4.  b = t
5.  c = c5
6.  for i in (c1, c1 - 1, ..., 2):
7.      for j in (1, 2, ..., i - 2):
8.          b = b * b
9.      z = CMOV(z, z * c, b != 1)
10.     c = c * c
11.     t = CMOV(t, t * c, b != 1)
12.     b = t
13. return z
```

Appendix G. Test vectors

This section gives test vectors for each suite defined in {#suites}. The test vectors in this section were generated using code that is available from [hash2curve-repo].

Each test vector in this section lists values computed by the appropriate encoding function, with variable names defined as in Section 3. For example, for a suite whose encoding type is random oracle, the test vector gives the value for msg, u, Q0, Q1, and the output point P.

G.1. NIST P-256

G.1.1. P256_XMD:SHA-256_SSWU_RO_

```
suite   = P256_XMD:SHA-256_SSWU_RO_
dst     = P256_XMD:SHA-256_SSWU_RO_TESTGEN

msg     =
P.x     = 03f6cd48873763fc0eb06947d0dcb35aea09599df5652ab3585eb3
          b0fe5dc5f8
P.y     = fda9972c0a5c1bb0b9ac1b1590404f7793d3523a194b6283c0cbb8
          da9f163781
u[0]    = 54b82282b0ca2fadc1d3aaca8b3749a6f626a0db55c1a71683e532
          1b0ca167ac
u[1]    = 12217139110a50ffb097ef02eec754aa0848d1ffc2dab783271e26
          2f525a5c02
Q0.x    = 06ecc4e1f2e89279dfd197abcd9452d2ad41686960cd9fdd109d99
          5e797fd5f9
Q0.y    = 43541719de000662feebe412bead45724d04b2b537c367144656c2
          5d70702fa2
Q1.x    = a16564b5afad9148200409307674e600e8624adb76e4a0e50a2063
          ba190268a2
Q1.y    = f3ffbafa60f93772e304f1f4121c98d2dcd5e80e2253eafe55cf8d
          cb3e60cf4a

msg     = abc
P.x     = f2edd21087e86cab1a9f01c8d05a3ede5936e641086ae07660b679
          178db1c6f2
P.y     = ee8df178c724d131a9ab173c2359a97e3fb7f70cb4060463e72b6b
          423c943754
u[0]    = a7b97013c7f7ddb79348ba163bbadd8128a770f425c96a16d3cd80
          ded548481c
u[1]    = 4d4cc838974b8d38b02b106bf96b4abb5bb8a78d0f8063911a7846
          c2c3144cd6
Q0.x    = 59035c27d2bf446222c7f42bcd67616b375f4f3738baa2d5386ddb
          0da65d5ced
Q0.y    = dc510ee04310d23333b4f385c01d9f39eaf0dac442165929b54dee
          176b47b810
Q1.x    = e0d4318efdcf2fc7cef4572945d5cfd1e65c89fbc29becd4871827
          81f98d0e80
Q1.y    = 0ed569711187cce5dd1047826cf266fa5c8319459240975ba02f4e
          eae5c3b4ca

msg     = abcdef0123456789
P.x     = 56d7548bd4db17f2d6eb666b1d119dc5a79087c66a3bd3dfc5232a
          99851e05d2
P.y     = 1d392bf473d962dfc82e413df0b70356a8d550ced314f397c8aeea
          1b1bd8b5e8
u[0]    = de1a53abd911ff4864136514a0a76f97a33cde47ca695cd659f718
          78b16e818e
u[1]    = 38216f3c99e9cc00602d92652e8149fcfe9cde6d3d742db9f8604f
          5be29e3a56
Q0.x    = e90130ab063fa8915f7b40d87dc844cb17615fbca777a8786dfa92
```

```
           0f7d64de18
Q0.y     = 3ed7e73dcb859ff29ee8c03c2fee155593e9bfa350108597823b52
           e8b550dddc
Q1.x     = 6b92f84f15e7ff8db05e5f6aa5d65b98786e5c01fc046f4080c3f1
           bbd175daf3
Q1.y     = 9217ff1c2a10aeb4e741ae3ea3502a1bc9e482844f31dd9c18d657
           fa79a9a994

msg      = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x      = ce6b06fd60933cdd9a92429d6a809a80a722d0b6242856e5d9320c
           173387e603
P.y      = 10a337579d7e0e5d21dabeb20bb447505898a26d4ae08ab9cc94a0
           cd3c6ffff1
u[0]     = 88fc351f653028d82517fdc955a47b7b7e0a7f7ba476e1c51d06fd
           87d43ffe86
u[1]     = bb4b5f971bb299e65baf86b3244905dcc51dd686ee5794a995e418
           03f9966959
Q0.x     = 639c0217dddeeab6b2994777cb203ac37c6d1ffc51e8c673ef5684
           f76e8f419d
Q0.y     = e3e36a9781f86c59dd808d5aa7b164a00b02c05fd84a9e84608a0e
           1d07d4b532
Q1.x     = c3f1acec963aa610b0d2b128bdaf425070c5ca6cef88a51f3316bb
           ffa5880fa4
Q1.y     = f36f6d9c0168659be9a0e58b168ae2e0e98189448adc9e88b75332
           8a900945ad
```

G.1.2. P256_XMD:SHA-256_SSWU_NU_

```
suite   = P256_XMD:SHA-256_SSWU_NU_
dst     = P256_XMD:SHA-256_SSWU_NU_TESTGEN

msg     =
P.x     = 2b1b6eb1989d3fb644a4d2962e5adab24444abf37963e3ee5c3a47
          aa864c0899
P.y     = 6b49f459088caf7588cde4650fba46f72870fe5bde98d170153bd8
          3d467caddc
u[0]    = 84bc3167da0939bc7130e749558acfa0001eaf26990b6f9565a882
          72edb162a6
Q.x     = 2b1b6eb1989d3fb644a4d2962e5adab24444abf37963e3ee5c3a47
          aa864c0899
Q.y     = 6b49f459088caf7588cde4650fba46f72870fe5bde98d170153bd8
          3d467caddc

msg     = abc
P.x     = 3a7c9353213d23c67286117b783863d0fcf0741df89680eb8678dc
          3482aa4294
P.y     = d023fd86e73e383327ebb25b1fc3980d0048c2c31a9927a6585ffb
          127f8b4262
u[0]    = 03f880f3c4b02f3afdfbb79f5a68f52a52dda11bb18bc615693cc3
          9e88eaef0e
Q.x     = 3a7c9353213d23c67286117b783863d0fcf0741df89680eb8678dc
          3482aa4294
Q.y     = d023fd86e73e383327ebb25b1fc3980d0048c2c31a9927a6585ffb
          127f8b4262

msg     = abcdef0123456789
P.x     = dc152e051787dfe673e1bfe48a72c5814c559e66c8e1a369084cbe
          73929d0ae3
P.y     = 12162397d8f3235e978e7aca400f78135f3de76b8f58c1fe4bf39a
          d4be424e0f
u[0]    = 6d895387b21ede1286f10a00bdc9543c2c33e3075287015ad50494
          86aafc8155
Q.x     = dc152e051787dfe673e1bfe48a72c5814c559e66c8e1a369084cbe
          73929d0ae3
Q.y     = 12162397d8f3235e978e7aca400f78135f3de76b8f58c1fe4bf39a
          d4be424e0f

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = bdff5810ac42130719752d50ccc5f79e169e90a90cda984c311f6b
          ad8e77fb44
P.y     = 73aa23cfb08dd1ed92d770152d59b7e2b63176ceb664bdcae8153d
          83609dd2e5
u[0]    = 91cb15a547813dad8255626afab8db2937bf9136e3e140a2281dda
          77568f0ae3
Q.x     = bdff5810ac42130719752d50ccc5f79e169e90a90cda984c311f6b
          ad8e77fb44
Q.y     = 73aa23cfb08dd1ed92d770152d59b7e2b63176ceb664bdcae8153d
          83609dd2e5
```

G.1.3. P256_XMD:SHA-256_SVDW_RO_

```
suite   = P256_XMD:SHA-256_SVDW_RO_
dst     = P256_XMD:SHA-256_SVDW_RO_TESTGEN

msg     =
P.x     = be9400f16f7a1ec3fb526872919f4f9af94c6ac3e8fe5787c0f415
          e9def825ff
P.y     = d14b462afed314ca7c20b1f6f3e501a610b79e0ca8adfce841b9c6
          804cfbe419
u[0]    = 4ad57c89ca3a6dbff76ce5a844aec126d7e4e58646403b3df08f32
          6676d8c8b4
u[1]    = d2be61ac4135806ff144acfd6d7f779a74b2cb9e91ff4491763bf9
          72fdd2e378
Q0.x    = 0e233e601d7e0caee9b4b48a546561aa257fa0b1fd49d62edcebb4
          fdddea46d2
Q0.y    = 498ef4cade20e4ba193e5ae5fa66d556f23d9293680c997e667ee9
          a409deee34
Q1.x    = fc0dda7c130cbd7f1e5ab25e6447f5c18f4f8a4c5fe7bf694a44c1
          7413c2652b
Q1.y    = a78e3f4af8b3f09f4bb126d5669dbda7343499161eac746a37828b
          b6e6f01300

msg     = abc
P.x     = 508fb4590784ea78f129c2eb2d67d73f6642f5ff15dd02931c477b
          cdcab87137
P.y     = 878c0c8b483ee6f0645c7effcf4ca1d2ed2f3654a527f7e06fbc43
          21cc15db41
u[0]    = f467e1dbcf8f75d17c272eb10f1d89213a078014631fa396b18dfc
          bb5a177e69
u[1]    = 210f9e593582b708768ed53faff7514fb238664acb2fb033188d2e
          9926e3f62b
Q0.x    = 1ba6f73dccbb69e1987bf676b5f51a994624c34874fcc96e4bbf53
          dfd8a8c2f9
Q0.y    = 3508cb3c41fe3c9c3aae84ddcacc2ec8ddec56c0f83875fedd87f8
          bdc6f54e79
Q1.x    = 8dac9d3d364c587e2bf43b856fe1102a2a82af80aa3f9e5593298e
          88f25ef2db
Q1.y    = a031a5907d7bd51c46b202c25cd36f06bb7c48f78bc04d77484b1d
          ef70634ad7

msg     = abcdef0123456789
P.x     = 7226cec474433278e405e2a472a0c44c27234fe5aa5507674dc762
          e7fde6f269
P.y     = 8c6424f339a34fcd937d7aa9234f8545bbed9b070e8f52f3893bfc
          21e121f0ba
u[0]    = 04adbdf1097d6637a52c955ba18f286337e3c7570195983a294aa5
          21d232cb1c
u[1]    = d9eebe468d34e0000da1de381f30e96caa582155b4baa0f078879c
          c045201ab5
Q0.x    = 2483744650307cf5c9b42d0dfeb5a6a552ddbc9f33306e26014aad
```

```
          49135a7661
Q0.y    = b42c2f1b332a23d294a395d18a9dd5e6076be31726a80ef4011b51
          a6e0a5d50e
Q1.x    = 34147e925ff536360e70fb6093dff185d11937eaaf03a7c27fe0c8
          347e0368ca
Q1.y    = 613746d811d1cfeb76632f0da3be81ef9ba86d4aa7459c0b21711a
          282abca3b3

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 34746e280053b2f059a749fe9bbf32bef6f0d1290c6e002245c5e5
          3759b4d52c
P.y     = 07a7235b1ec33076a2ec7a8478af55f39c766e69d9e0fee8cae327
          5276c8ac04
u[0]    = a0e77c3fb10bed3ffb1c154877d3dd327569e10ac93d186b2787f1
          5095c59d47
u[1]    = dfe760152ea428430ca0ddec8ce33c9657b5d54efcf1f55d861910
          671efa3fda
Q0.x    = 85381155e57753630c0bda6f012ab819f2bf309819c1e9c20bf186
          a84705bd5b
Q0.y    = 173b8e141b32dee991f7498077847b0f76a61125bfbc41e9d804fc
          55caba8abf
Q1.x    = f98bec489e4d2c5d8416e0f1bc40510a8169dfd91ff06ff4eece76
          f44392b8a5
Q1.y    = f165f1b2b2cb47a3fdab38fb41d8947a2c565e685737c8ab6ec095
          14d4000f4e
```

G.1.4. P256_XMD:SHA-256_SVDW_NU_

```
suite   = P256_XMD:SHA-256_SVDW_NU_
dst     = P256_XMD:SHA-256_SVDW_NU_TESTGEN

msg     =
P.x     = 8501dd0a6f2313cbe3adea8991bc7ce09400fe0ed9fe4ab2930bf6
          325cdb35f1
P.y     = fe28c7d2476ab1464e7b097d77ad527c3ef7265759bbb01bd690f0
          6542beb4d5
u[0]    = aacf70475184d6532779d2e7b1b148130dd280fdf07eb82bfb3bfd
          2e61d5b441
Q.x     = 8501dd0a6f2313cbe3adea8991bc7ce09400fe0ed9fe4ab2930bf6
          325cdb35f1
Q.y     = fe28c7d2476ab1464e7b097d77ad527c3ef7265759bbb01bd690f0
          6542beb4d5

msg     = abc
P.x     = 2873efb7a560eab63c93e1e35e25c4867d9565279b2954a72f7e68
          e5cb398714
P.y     = ae5f67e81613340d0e19238c055d51dc181e740267acdc85285608
          00fb5a1661
u[0]    = 1befe400c70c6bd32d63d8049db60afa9a19cf4718164336359c0d
          9891cd6bbb
Q.x     = 2873efb7a560eab63c93e1e35e25c4867d9565279b2954a72f7e68
          e5cb398714
Q.y     = ae5f67e81613340d0e19238c055d51dc181e740267acdc85285608
          00fb5a1661

msg     = abcdef0123456789
P.x     = d4b392897e9f709a1803314c04f34ff29030400aba9886e254a45e
          061cee5059
P.y     = 5ce9455cecc9a105f73ca63f068a9bebd5be12705bdb5ba2c5edbb
          692d48c2a4
u[0]    = f49957bfa6cfa9732f1094e5ee12df073eb0915acdf5fc65bbd1cc
          796edf7b92
Q.x     = d4b392897e9f709a1803314c04f34ff29030400aba9886e254a45e
          061cee5059
Q.y     = 5ce9455cecc9a105f73ca63f068a9bebd5be12705bdb5ba2c5edbb
          692d48c2a4

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 54e8999525538c4c067ade8e68979a4299a0b4e11328a1bc7a7849
          7bd007273a
P.y     = eab90502eb73fc640ef054f108cb40c9e89bbaf3b1cbe35735200a
          371e823e90
u[0]    = 68dd1e91ec38fb8638dd718d82f437658b815c7f57a74ca6d9afcd
          e5f18244cc
Q.x     = 54e8999525538c4c067ade8e68979a4299a0b4e11328a1bc7a7849
          7bd007273a
Q.y     = eab90502eb73fc640ef054f108cb40c9e89bbaf3b1cbe35735200a
          371e823e90
```

G.2. NIST P-384

G.2.1. P384_XMD:SHA-512_SSWU_RO_

```
suite   = P384_XMD:SHA-512_SSWU_RO_
dst     = P384_XMD:SHA-512_SSWU_RO_TESTGEN

msg     =
P.x     = 619d4168877421106aecb16ec35d84b7fdfa215cdb446d82bac49f
          6eab51a34b4d5314823f7639293cf6471c6c981a99
P.y     = e5035c694665ca25b2c57542673af6b91288110b0b0689657cd031
          96976d82dec104fd9f91296c85d1ed94bc9309840e
u[0]    = 091d2a7c61e98f914380f54b5a19cdf32f51aecd11e65ce494112e
          e6e4b3b4a6298a775a4178dea6090874eb96c689ac
u[1]    = 3b2fa6330a16df240bd84c60e815346d06511f55969f6d4f8c26f9
          9854b72347892d8a864c16ba8df16528c52f290578
Q0.x    = a2c93220bc6918cbf2124d614ce2edeb7385b4e822f55cf596d94d
          e6848bb67c6e4e7a70ad85cd2bb9cb1496f4dfdc37
Q0.y    = 70184f23bd4fee377a6708eef5ddecb15333ad40a4d28715233168
          7054b803efa89685605b40866cbdf9140dffb7a90a
Q1.x    = ba5b9c205ce4c2da6bfd0a8b0d2234d3f6318fb2565c719a0cbbc6
          7626645932500a915efa0d29f0bbd10f248a5e6642
Q1.y    = 4f263dd72c7741e8c03bff484e306001ac5b31df3ca151f025c7ad
          791b0f453f7d3614bd1c24414e609c06376d1fe498

msg     = abc
P.x     = d7c33555606b86c3ffaa1a645f806bac9d553a769f5a735d75a395
          d58a70956b6d3bdbd6a6a8c83121678a036005208a
P.y     = e1c55f372a905040576f61fbc07e9664359e76f3e7b5be8dfe7224
          720f85753a823e94a3f886ced2ec5ce13b1248147a
u[0]    = ffc8bb0c99882b5506c5e7950f33dd079cabf702a9e04fb040a810
          ebc8d40cf825728c78dbf1230c480e2cc371f1d4a4
u[1]    = af12b4191e814480159b520a789666d00c99cb29344569ecb221fb
          8cad69411f0096775eaf1f4face1edc85db658bb3b
Q0.x    = 3a0d5045f51ec19eae02a59405b4f5f3012c01b5aa87677e0fd35b
          bc08a1316562207ff2d313282e6524a714a1fec282
Q0.y    = 9aa73f6108e6af8faf5bfec00ffef05ea7559f8100c67e997e00d5
          d847d9b98f2461ee3fd303341d9f02f7f1b4f4a9de
Q1.x    = 184296dbb14ea7e17359be458e801bdcd37bfb4d259c9b4bcf88e3
          7ad88368fa739dd26ba0d2a632ce35ffdd35176b78
Q1.y    = 73ab89802e651825e77a99361bb2cbd365bcc5301207e99be0a954
          d49628dda943dd0c3f41d8775405c987bff62486ab

msg     = abcdef0123456789
P.x     = d1f4bdd7ef9ee1c2d57ccd2b3b80123ccf3eb64b2f0a3ad26b8cd1
          a8a8e411aadb9922d0e66a89ef0e78dba1489e23ea
P.y     = 9f376abd97ab8838c604a05ad17be1dfe0d924ddf0184341ec8e5a
          ef9efb0f6559ab3048b4e1e0e42ac19ccb6d1dd892
u[0]    = 5ff66529c7354d61088a998c43e1e2dc7603e40d4118b3f8c0f842
          6e12b141251d1aec9b5d597d01ef7093199ed0c67f
u[1]    = 611ef2efe2da7beb61ed2158e2db03f06fb3d4ce485c7e30575920
          07a91053c22d496ba112645f2145d4b1bd7bf98a2c
Q0.x    = 455d7359198a678bf4de0a6dafbe7929acb93e7dc5b50a260e5998
```

```
           0828038247c22a88b73a517bdbcb0b90671037ea84
Q0.y     = c4737acffcf31b97bccb55923fd5629d896a636f50f46ab01aaf55
           647334bef77164861dc3a7af96a591230a40a41f19
Q1.x     = 6836e44666701764cbb69216568c7c52afb341f294b221f1dbb646
           fcfde97c6650ae68c1e69f9ede4c360d79ff7c85e0
Q1.y     = 0e0bdd40fbaf98dd8ce0e3b32ac61113369be3600190ef8013709b
           961cdedb2f5273873aee848d48d74a07e7f10e6022

msg      = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x      = 1b669293d6edfd55ca7f9725d72fc62b55a8829738404a1239f2ca
           783ad8f8a3ebe2ce54a28fcb6e9eca85d1ce29a929
P.y      = 25556bc2c6382c261feb362519f1e5d616518810262c358fb80e45
           803ebcd0ce830b594da7a0e9de9eb13ad2d9191d34
u[0]     = 632014cbbab75556b66ef27806aa016c170cb5f7cdc7cabda43db4
           0be1323faacf90113acbaf4065fd26552f085026f6
u[1]     = 86e70cc3af0887e8fc4b5216405fc3fe7277a1e43d88068ca74d8e
           4b6afd817ec14d4b57bb7de29d71ddf587f73d57e5
Q0.x     = 88672e48d6fb2e5361e3f550987d74fefb55083f826f9c45f4e601
           3261750cb533e085c99993dacc87702ebbbceab269
Q0.y     = 18baf7353c64daea70b6745216865433fe693c63da2c7a3a72aea7
           f263d0d8a9274795b0fdb45bca9ce3494416d1db18
Q1.x     = 4b76e5d9a68727d35f90d21bb1b2425b20f8585aa740491749ea76
           884f7f5b2b7df3d3d2dda4f520b7b1a321f5c71acf
Q1.y     = ef067ae93f7a1e7ffe39e47e02038a6571ce53e17ae166555f0228
           1fb1531c031de3a43f6e614cda04c2f4b0e23b6783
```

G.2.2. P384_XMD:SHA-512_SSWU_NU_

```
suite   = P384_XMD:SHA-512_SSWU_NU_
dst     = P384_XMD:SHA-512_SSWU_NU_TESTGEN

msg     =
P.x     = e301de73b2aa0d7d10916a1c96fbb5b257a763e07af63b5070eeae
          a5b5802e95e8dbe2d781bb2b071393d7076dd05ddd
P.y     = 54fc88309ad1877a0161d3306248f79b740f10f10006b05a47f81a
          163b065258ed1323a35eeea5cb1ea86355ba31b9f9
u[0]    = ec21664b5b562ca14c6e3d882e1d688a6565985c8486af3c4d8951
          8ce283f9f2b7cc8a5b55968c62f85b420678821e83
Q.x     = e301de73b2aa0d7d10916a1c96fbb5b257a763e07af63b5070eeae
          a5b5802e95e8dbe2d781bb2b071393d7076dd05ddd
Q.y     = 54fc88309ad1877a0161d3306248f79b740f10f10006b05a47f81a
          163b065258ed1323a35eeea5cb1ea86355ba31b9f9

msg     = abc
P.x     = 586d7162ddfc0233b8ba30d29ab4fea344336d978f771cd991e1cf
          66b3ae4d29c0c01c490bb2073f9afbcbe16190a2d7
P.y     = ed4e34ea44b6e32976810f9bc96e9c8a5bb5890abdbcd59283e11f
          9b7c5089f94b46758db2add2c59f0ac12a19d9d795
u[0]    = 3d083883a17c60649cd759dd61302da3d0cdc9e7fffa7e1dffc013
          583890b27574473d35b7b8f90a63bbb3e38776bedd
Q.x     = 586d7162ddfc0233b8ba30d29ab4fea344336d978f771cd991e1cf
          66b3ae4d29c0c01c490bb2073f9afbcbe16190a2d7
Q.y     = ed4e34ea44b6e32976810f9bc96e9c8a5bb5890abdbcd59283e11f
          9b7c5089f94b46758db2add2c59f0ac12a19d9d795

msg     = abcdef0123456789
P.x     = 86114b3da1e654ccbbe442d2ad548f68b19b8773e9fe4d98ea04a8
          adffa0bb63337467ddf04df926a07c078a444a6efc
P.y     = a4505d54ee92518dc640792e7a1b07222dfcb61a3a6ab319b745a0
          97c42ced72f46171fd342f96ce72aa6224f56cd01a
u[0]    = fce089572bf8eec0b94d8c619c1a198dbb70ed129b3b7e22351a14
          defa02cda789b15ef3c499c464188140bfd0d78f10
Q.x     = 86114b3da1e654ccbbe442d2ad548f68b19b8773e9fe4d98ea04a8
          adffa0bb63337467ddf04df926a07c078a444a6efc
Q.y     = a4505d54ee92518dc640792e7a1b07222dfcb61a3a6ab319b745a0
          97c42ced72f46171fd342f96ce72aa6224f56cd01a

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 28c7dafb30a83e5cf3c984eeb191ebe742954f87963c91fc717b9e
          bf3880cf40837a8c5ccc734846ae98ad6096462459
P.y     = 32e3251f9869e546ef37311696fa047c8e45e60a809de166ecda3c
          23d95cf8532248dfd87e4e50b3eaa1e3d704d5ce53
u[0]    = d1f73f4b4f1d8446aa0cf6e12f1737e52de386a16dad52d3af1fbb
          320151b09d70240979c346c6e15f2c9ae6a734f78f
Q.x     = 28c7dafb30a83e5cf3c984eeb191ebe742954f87963c91fc717b9e
          bf3880cf40837a8c5ccc734846ae98ad6096462459
Q.y     = 32e3251f9869e546ef37311696fa047c8e45e60a809de166ecda3c
          23d95cf8532248dfd87e4e50b3eaa1e3d704d5ce53
```

G.2.3. P384_XMD:SHA-512_SVDW_RO_

```
suite   = P384_XMD:SHA-512_SVDW_RO_
dst     = P384_XMD:SHA-512_SVDW_RO_TESTGEN

msg     =
P.x     = 116b610b993a485a113916f5be8bb682263f8a2110484ee985c8cd
          5c11ecd52d617742a2f3c16dcfb3bb60de53151ac2
P.y     = e1fb2f2de9297eaea87b6552682b04c59ab5419b291477a85a26a1
          56a8a72f596fae14c5b1188accdabf6f99664b3e98
u[0]    = b22b20d1c078b3fd21c90105c7aecc0a4ef5b6a9faebfcfd6cf6fa
          892e0794b7b023eebc3c3998cbd77e3b12ce59dbcd
u[1]    = f8536be13271b8fe5f345d4f06d8dc6fdeaf1b489bea07ecf1c516
          f45859c049fa9f412a39740936fd66bce4ed9ca85e
Q0.x    = 02f7273b142a6e1ec112d31bde5189992505a0792a5fbec37f2c2f
          6df0784ce383fbd787a6e0f83e3fb44e1ccf35d2f1
Q0.y    = dc43d0f30ce6bd01b82946687c11d37acd20688477dcf3fa617d7f
          beaea7b4f730bcff5ea6e3e58ecfbee3ecc17749a7
Q1.x    = 9de548b51a0afd8ae7d3b4a0ae2d2ec988d3ecf74d50de65931cad
          7be828355737be3f5d72d61b21322171f2ea9edf42
Q1.y    = 6a40ade6703e53f4ce39f9fab4c8b27a82267c4dd6618ac8e8638f
          18ac816536517c1680fc21efab8b5fd4fc42724cf8

msg     = abc
P.x     = 544796d77562735ce9e993225f913b8202af7595cb06e16e62fc4d
          a56f5908994cbc5dde35a91595586515b26b6ce4ca
P.y     = 7fcf56951efbfc565fe3b4ead037a92e20d5bb03622f028db59355
          3fc1d5d7293526f742fe26fe0a38fe94a2c2c4f1ca
u[0]    = 79a4d89cada964929bf78f77facc181ffcacefef6439e1443a8640
          b88c212b578decaeb2b91c8915be7f9762fbade780
u[1]    = 2bb7989a70f32b3b8bef707a83ad822a71f9c065322aa604f0d1df
          c8f576dee2f3c94619a9b9fadf2f739510021dc46e
Q0.x    = 2f72995c4e9d8f660131ff1a2ab03041632f06444c6084ad4c3424
          a4816f508d060c3a85d34b885ddf77955fd5917df3
Q0.y    = 2d5a3486cf641336a92617d4a22a502dfe8791b13849488520b309
          f7fb1916b5338c24a883147bbfcd34093244efc788
Q1.x    = 2112ebec9794ee455a30d3dc1a7e00b4d48eda6585ee5bb6b7c569
          96a1ecfbdcbe744d3da5c10965422f4f979e69caca
Q1.y    = a3cb85d19867d48195da52305b0e02dae32c8a396aa6e28f25edb2
          4e996953bbb6915065d93b9908821e62fe067a6306

msg     = abcdef0123456789
P.x     = dd25d11fb04cce44f87a8eabb47dfab3223f967b485995de7f733e
          cb2cc9f74cd9fa492fd158c7587d5d0e1aea9c4b15
P.y     = 332a94ff5101e3cfcd55a1449e8a99376c5985ece155079e498085
          4ae9be1c2e93d5d2643600480d7b693046bd2b4b79
u[0]    = cd1ef357a816a8e353c6cf37023067226bdec751b30855abc440d4
          fa7ab6e65a0c20ce04fa1e4af21e40725d6705aa87
u[1]    = 7d9982d89b869fe5fd0712be092f426c9ae4d92d9c53d6c8955310
          dd6ad42994885e2cbc51676f33d5893410a511e8a3
Q0.x    = 2799a36673ba6c950da6175633c2abb5d2880461833e0ebfb4fbac
```

```
         3f83119dff7c316c5aae03dff253f28e4366be65aa
Q0.y   = d7aaeae65d98df66ba758771653a69d59e13d623f04731dc495cdd
         80ad588501e4e8f9d35ce654819a8d92a646576b21
Q1.x   = 5bfb20a562b5c02e5eca70bf9b9b2b1d9c47b68f55bf26d42a4bf1
         741413262a5c2a43ae1458b98d792703b25d418fc3
Q1.y   = b41797642a62d717932a257c3500a08c8cdb19da402354b85a7288
         8c7c68d6fd0c8821c4fc0aa0d0551a01e7433b4abb

msg    = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
         aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x    = 04cb2eb225d6c0fca3ab45a3cbfc30b45202d3cca0e79526061d10
         dd8553e50b82f760865d62e6cdbbd29adb1b299e79
P.y    = 1e22c1c3e7737f3a6b5075da1ddccbf9f85d466bf4e07798bff3df
         75ffbb2bbb6b28424a081a52020e191133591b83bc
u[0]   = ca87d41db1ef851c1414e156e9cdb12944afe8538953880375e505
         4aab81a63514a391b62f43bdaba56e547a5da12522
u[1]   = 781cd222de78756f4d7ffb6819e814fd1790ce6771ce208194e34a
         f537876be6d5dd2c0a9ff9b8d60319dc8c58bdc021
Q0.x   = 56cf2041d2afeae7545fb360c64d7273f8783c420158523639b9bd
         f2a7611b5d97081cf420dc3f440a840bcdcf4cba93
Q0.y   = 0cb2f1578dabd7a7ea0a076921ee0928e5b486dec806ca245362df
         6a96d8c78fc3d4f1a5387ba56ab4c8c02fef3f28ba
Q1.x   = ddde570972e875a0e40e6e6e277341d38a3a0af0d52bd9575293d0
         d9a90b7f2c60b1568c34c1b0720369203e580bcec1
Q1.y   = fa7df2fb5c0b9e462b237e75749842ffadc43f0cd57fcd9b863716
         a1aecec0cef768562af664d10bbb92008bdb00aaed
```

G.2.4. P384_XMD:SHA-512_SVDW_NU_

```
suite   = P384_XMD:SHA-512_SVDW_NU_
dst     = P384_XMD:SHA-512_SVDW_NU_TESTGEN

msg     =
P.x     = a2c7fd2c9097273267214a8031b0fdc7c5e3475a08ad27940ce6da
          329f55ceeb19046cd9f3f3dc2043ed1e0633f848aa
P.y     = 4c6d2e0a28bb0b2c68c4116a11896b235eba951f222e7966b86e3a
          709df700d10a338eef31180cc8ed084d5c962d04c0
u[0]    = 63956551cf429ada10d0caaf0834dfc7147bac3cafe8562b3f8327
          5e62799ce55143e378595034ed969e971d0065a078
Q.x     = a2c7fd2c9097273267214a8031b0fdc7c5e3475a08ad27940ce6da
          329f55ceeb19046cd9f3f3dc2043ed1e0633f848aa
Q.y     = 4c6d2e0a28bb0b2c68c4116a11896b235eba951f222e7966b86e3a
          709df700d10a338eef31180cc8ed084d5c962d04c0

msg     = abc
P.x     = b0c82cb3343daf4320712673f1ac67f4d14a4ae8d040ab5529952e
          4431dbcf1b2cb47a3bfb16956857a48a00a62dff6d
P.y     = 94a1e3da0b75370b2a62d0c8bfdc7996e675efe0e65ae74628c929
          78b18ba3cd075550da1887a7da394492c7dd2506cf
u[0]    = 84f5786378e0ac4ce30a349121f7d7fdcfba9d8f4549ef2e33ce53
          d68cf0196302c8b426fd1ae817dcea42c59c82c2e5
Q.x     = b0c82cb3343daf4320712673f1ac67f4d14a4ae8d040ab5529952e
          4431dbcf1b2cb47a3bfb16956857a48a00a62dff6d
Q.y     = 94a1e3da0b75370b2a62d0c8bfdc7996e675efe0e65ae74628c929
          78b18ba3cd075550da1887a7da394492c7dd2506cf

msg     = abcdef0123456789
P.x     = c7b19f3fc636f2568206ede8b74811dc7dbf0bba449dc20fe680bc
          7548286d7a31d2b981122df38bb3a8d80571d71a40
P.y     = 4b2148246465ab125f7d31739e5b675ed5d16de310fb3c4cdf6999
          cc3867dd525c87fd809e580f62d94a19ab6f4ff2a2
u[0]    = f614a511e6b8f03c00c7c82eb4c399f0e833eb7c01c9c36c00b642
          4c37c198c20276637f1d2569f40f229930fd9de2ae
Q.x     = c7b19f3fc636f2568206ede8b74811dc7dbf0bba449dc20fe680bc
          7548286d7a31d2b981122df38bb3a8d80571d71a40
Q.y     = 4b2148246465ab125f7d31739e5b675ed5d16de310fb3c4cdf6999
          cc3867dd525c87fd809e580f62d94a19ab6f4ff2a2

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 277012d7c8638c070e7b04672a3e56c4150acc31616461d6429b56
          e21f8824952e0194fcd2dc9858a4f915dc3dea61ec
P.y     = 2f59cbe11b24e0c13c5318ef52438576d7109079ca0590e923cdbb
          76de4f692ff250cc46a4cf06316804e680f964365b
u[0]    = aad065adfec6745b83223ebb23f9769cd8457ec1857479d3d01eb2
          ac6f709a5a19033d67373d30f3054c3a4f43f94ef3
Q.x     = 277012d7c8638c070e7b04672a3e56c4150acc31616461d6429b56
          e21f8824952e0194fcd2dc9858a4f915dc3dea61ec
Q.y     = 2f59cbe11b24e0c13c5318ef52438576d7109079ca0590e923cdbb
          76de4f692ff250cc46a4cf06316804e680f964365b
```

G.3. NIST P-521

G.3.1. P521_XMD:SHA-512_SSWU_RO_

```
suite   = P521_XMD:SHA-512_SSWU_RO_
dst     = P521_XMD:SHA-512_SSWU_RO_TESTGEN

msg     =
P.x     = 00ad6cb736cb0565a2b6c52dd9e53f76a9a40a44c73bfaacef03c3
          ef62a9a23920b7df4de1b92754de7bb3013d9d36049da001136e7f
          4b1b0ba10beac862a2b3d3c5
P.y     = 01c2ecab1b6f7bb6797a4b5bd416b385e891926fc17f230f2406f3
          d47076526c5d90bfb4d0170fd8a339de1a66e6304d280d0404fb68
          5b2ca07e2742a770b681bf56
u[0]    = 012bf001213ebf5c37474cf09f804fdb239c82edcf42d1eacf54c3
          2e3400ee2c4e6d8231fef3a11633b25f8a8c4c4e827d0b7db16c25
          67bf5a41413d0218ec5855a7
u[1]    = 01f06e1a4dfeeac6810422d5da691121f75d1db17e187f65b15550
          ae6698bb67173544c28a6d4111691081a619722f70840cf2665207
          e82f35634f379351af0d4637
Q0.x    = 00eb222cc5bd63364f650c2182a52af6deb789783f6cc2b45f67d3
          4a8a238f4c3becc371d1fc78d29697df81933e62939c805c384115
          e66733259a3f7145ca8085f2
Q0.y    = 00a9bc815419728122b2503b3f96093c33a16e2f9817eb32870db4
          74e3a6411e5f8c09b32de33f125732bc3e1515361abc2dd74ef36e
          ca66af95fdb3fbef8f149cd1
Q1.x    = 008e76a7377126972a4645e17c3930a390363a4073b6401d8ce508
          68e03ab449c7d433604c014c255989391a537d86b99af867d87297
          20d08185bfa0df440b7aa923
Q1.y    = 0072b4676f1900568f7d6614f4661d0df86d9564936f92eb05936b
          f9c3dbad1019a3588893b8ea98f71ab8e587ab38cdb807f519379e
          abc7f8d27dcafd7b130d2843

msg     = abc
P.x     = 0141b07880eab4a77b7ba29f0f0e85fb202e6b020ce1b5395b54c2
          50d62f1ae770e9f0315ae70edabfaedba56803073ff885da8c6fe9
          ee62eeb81f810caeb88f528f
P.y     = 011ed75d5f62abd63415d4db3c7584ad743aa89e1ff5c1b250cbb7
          2eae6e238e5e486feef5f2b3202d945d181e8108109d9b5dc10047
          672dd292b25f7c1b9a5541c5
u[0]    = 007bdc26e59c48e161d5805dca3929431b370e351f36baf8351726
          0c4ed440eeba12a4c34209d74cbffb2abdc0fbe471a79d9bec6cd0
          af536c41445cc7b665e84534
u[1]    = 0134b98b8e73587124752a418b1cee287ecd410a7896f32815ae23
          8a5c0eccca3eda271ae4c3b23276c7d211fb9276d77e6ebafc5071
          4d0b403c9e9ba25f67ea3aac
Q0.x    = 0112888dbefe4c6b546fba832693c1030585d2109820a4f6f8863a
          b51f33b80ac1382169b196039ad4c934674a962599a116b0b7debd
          e24cb43953d02eb845c07c1e
Q0.y    = 0019c0af8926bd00b1e34073673074b8a6ebcddae3643f2b7e5760
          b010c84cd3589d666c58a5d2ae214878cef1e9fe630b3d7b07c025
          3eb71789b7ab22987fb2794c
Q1.x    = 01455e7a85dc202348ac075de68a45697675d7086d1702ab8268bf
```

```
          27d5113e94bb689ac338c486d6f82c8cc938f89c8cb7f07eda36e7
          be906798615e25f4ce73b5f5
Q1.y    = 01cb2e858cfba8c1ad216fab1f535e7a397a5c83c36789d568cfba
          83c4d7aa9333c87d608fea72c37d3dd1198d0c7e1636a699bad6ec
          6d794b95aca76dfeab313f8e

msg     = abcdef0123456789
P.x     = 019f0195e514da4243a4d2de4b7ed2415d5205c6da11eb7deae70b
          e78a61bb89ebf17f7c9970ee20b4152ae50c95e55f626bc7350d5a
          0f530a91f48047bd90eeeaa7
P.y     = 016eaa02cd5511a96eed4ffc965bdc3f1fdbb7f4c9895eabdf168b
          44250278ebca55474bc89a2f246b8fb959010502aab8a9385319bc
          f69f74dd8f518bea1c7fafde
u[0]    = 0117ea59f8ba7e6a8d4c5b4337c6f34c1e0d28769d5c5e1c73c8f0
          9c8386e7c8b74bff5158553bf0d8ea1ab45e4e61cfa4e0893a7061
          562368ec11a8566557fe75d2
u[1]    = 00431e5043c40e56afd165e733f2280922f23a31ba0b2e12ffa130
          4ca3fd87fea0b309e90bc3e0b9e03f5198120e1e0c3ff56cfabb8c
          49f35460cf495a72f295e870
Q0.x    = 000d0164f2e5d7748f33824344af8752ca4741a9d045830c49d93a
          debf57550253b793f2370290fc0b22919afc18bf663e5118232059
          67f0aa94033302244d83618a
Q0.y    = 015c8a9ecdb5ffa0a3949d2004dca96157edc8804960bc30b17a52
          6d9dd09be3d6ac1fcae0d1c9581eec9d99e561737c734fa43b15d8
          4e3d194cba74145080d82d5a
Q1.x    = 00cbde7f4491d11480cd1140e4f7439745d6f0aedcf0ee721228fe
          9a49ed8a689aaf0463691ba499e7f609ef56559835f85978c201ff
          0a457cc9a72ff4cca24a3e2a
Q1.y    = 004dd85f141148bc7cdbd0877d8d195038aedb3d4708d579aeb4fb
          f94d8a1c81910d0d00d055d2ea18c4df0ad5fbb43b9007603aa4be
          47a5348554aa6a2c1ad39dfc

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 00de3dd7780cfcd538f7b3067d8da52c522a031244dc0d327e6e99
          ef331be475354a0b481a22e0d4d35b232da260baac5b693a827a20
          b1f328a416ffc47ad945a4dc
P.y     = 016bb6c5c965c6a80a4a5e0c2c7bbd841766eac695f88a730076b3
          2d4399da01609a4a17b59a21f4f58a174d6110081b96e5aaedead3
          cfd4252e74de969680ba74ab
u[0]    = 007cc5f6f0cc942b7c6148c3e1ac54553f4dc3032c59e5bd862bbf
```

```
          f6550c656597283db923673152f5c92b0fa59f38d433899678c09c
          d9753adbf3f5c4c5c98d0764
u[1]    = 01ada0f87b5b64db9d187a2a9e0d802432891289bdadbc8c5a18c6
          716dd8820ebaabf4c5f05f67f4f5febd2f4ab761d90b1f951d48f4
          2cdd11eba88ae114dd554b5d
Q0.x    = 01f0d615b1d5e2bcd0c5aad6103c45d8c38fd248d06497b79bdddd
          f25c37adccb461439122b4c59cd689ce534083b56d2f5ad8b66eea
          ce78d95038e77e278b619f71
Q0.y    = 013e1e9a917b0504c79e6e0cc022fb167ec2a9bd479cd490639b17
          b0925f502191a0cffb2310344da59b1fa331e8e71775c6426c8735
          9a68d755c7e2556e690de872
Q1.x    = 00b16209974a7b2719d08648cc811f70c029945d4b17b5f26c333c
          8876c7e21759468de547e12ee0a4eb04cb29d5e87209d71eb0f8a3
          e5bc6169bc4b2f8d8a59de38
Q1.y    = 015a551e61b60bb10183089d6a1355a8f24c082aacdf9e2e9b23ad
          ff36598b13840c4a848453794a93e7e5f63278be6585d9d62e271f
          f8c0f5f16db085abadb7cfff
```

G.3.2. P521_XMD:SHA-512_SSWU_NU_

```
suite   = P521_XMD:SHA-512_SSWU_NU_
dst     = P521_XMD:SHA-512_SSWU_NU_TESTGEN

msg     =
P.x     = 0074dff9ec0371e4de820fc4b2a8a5e71498434a458a4805ba11d6
          2ac80da2049fbccc3e4bf20a45efcd04344528fc1cbbfdbed6b2e0
          5c5cc3d4fe55e00fb2647dcd
P.y     = 001cbb63a0161e5eee698ae8044949c00c0164fc247f3f2a4ecf2c
          6c0a03d5a64593b563623c1a8aee92ec23dfe2430fa5a912a825ca
          638f4b13963cd14729424b57
u[0]    = 0193d358a22125c3d2b7515b0e9ca279d1089ee0a4c806e5d585ed
          585beef9eaae37e0bf2c632c60e94fcb76db30418e7afd9c713bf7
          d47ec8ca6b851a5d8b69a98b
Q.x     = 0074dff9ec0371e4de820fc4b2a8a5e71498434a458a4805ba11d6
          2ac80da2049fbccc3e4bf20a45efcd04344528fc1cbbfdbed6b2e0
          5c5cc3d4fe55e00fb2647dcd
Q.y     = 001cbb63a0161e5eee698ae8044949c00c0164fc247f3f2a4ecf2c
          6c0a03d5a64593b563623c1a8aee92ec23dfe2430fa5a912a825ca
          638f4b13963cd14729424b57

msg     = abc
P.x     = 01a5084b5e79f60471b4cf3524bcf441a5be3d6ef10be5dd534ed7
          ddf9c093f6bb79f53792b7a2e38c3610245c69b49511cc4b882551
          048a5c2ada5dbf81be0ae471
P.y     = 0162d8cc7e11ba34475e40e8bb6bea8034840dc82f72338e9873fc
          8105921dbf980743927ab8476610c9983b158a63d82d5c18d71df5
          98b325221bfaaa82b62d9c10
u[0]    = 00e349a6622102a38650a7405fc2bdf1245b8ea21e03e76060c23c
          fb2f6067522026aba99a4bcc8e8f739d1852fdaf5e51a8114aa033
          ee9d10832ce1084ad102dca0
Q.x     = 01a5084b5e79f60471b4cf3524bcf441a5be3d6ef10be5dd534ed7
          ddf9c093f6bb79f53792b7a2e38c3610245c69b49511cc4b882551
          048a5c2ada5dbf81be0ae471
Q.y     = 0162d8cc7e11ba34475e40e8bb6bea8034840dc82f72338e9873fc
          8105921dbf980743927ab8476610c9983b158a63d82d5c18d71df5
          98b325221bfaaa82b62d9c10

msg     = abcdef0123456789
P.x     = 00e823a635ca827853994d748f78f407cb99ecc16166da410c30f5
          b728ba5d9da6c1664a7a123d18c538b733e54e4bb0e9bfe743ec62
          60debf547cd61af94d8af6dc
P.y     = 0170c8f2d8e642e1651ca96c10b523f65e3f982945b6698effc78d
          15e148d0455ed2370a3fa898613c9407e454db1c9ff39165e44a02
          cd2397a95c5f3f102599fe7b
u[0]    = 01f402cd45679ffd7adcb768498a1169fc2906701cf7a2fb2bafb6
          7a6a2bcaf426159995c1c5cf135c92e49ccf94e9534c5b966ea51a
          ad9a6a9747b44dbec6e3ae99
Q.x     = 00e823a635ca827853994d748f78f407cb99ecc16166da410c30f5
          b728ba5d9da6c1664a7a123d18c538b733e54e4bb0e9bfe743ec62
```

```
          60debf547cd61af94d8af6dc
Q.y     = 0170c8f2d8e642e1651ca96c10b523f65e3f982945b6698effc78d
          15e148d0455ed2370a3fa898613c9407e454db1c9ff39165e44a02
          cd2397a95c5f3f102599fe7b

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 015ec57e0b995da9568e8d2b1b3856399f24ff650f4daaac8620da
          5613fc8aaf011691744db8ce27498b6d9d313952ca00dc05564da7
          84712fb4bc289a934940e16e
P.y     = 00b422a44a249e3c4d2b534bce9d2aba782f7d81679b786ab4f5c9
          7a34702a354851d7cc5f21397be3a4f9a83dcabf1a3241255c4584
          0b068567e2c843607ffe0937
u[0]    = 00deed8428df4b4e18889cf80998fa56f34c53a99171a7ffc6cc22
          693f89c13a13713e4a0258839ba192c1fb1cebc97ac66e8eaa30b8
          f5e747122aa35df669b240cd
Q.x     = 015ec57e0b995da9568e8d2b1b3856399f24ff650f4daaac8620da
          5613fc8aaf011691744db8ce27498b6d9d313952ca00dc05564da7
          84712fb4bc289a934940e16e
Q.y     = 00b422a44a249e3c4d2b534bce9d2aba782f7d81679b786ab4f5c9
          7a34702a354851d7cc5f21397be3a4f9a83dcabf1a3241255c4584
          0b068567e2c843607ffe0937
```

G.3.3. P521_XMD:SHA-512_SVDW_RO_

```
suite   = P521_XMD:SHA-512_SVDW_RO_
dst     = P521_XMD:SHA-512_SVDW_RO_TESTGEN

msg     =
P.x     = 01150883968dacf7ce208286699234a0bb3f791683db216b768a07
          1ad25a5246fee1f57bf9744eef50c35115743e07f639f202d2d571
          25720fb78d7c89291e6f59de
P.y     = 017407011b5d19aca70a0c074ad6e4479aacbbbdccb1483a03b17f
          71ba4d8bd2dc805e673dc380f948602de7a8914664523767156587
          3d94c33563bf8f7a58883e3b
u[0]    = 0086d686b9851408cb2dcef6671a6e6ead3b7ef0aede006c365b15
          db2f2a63f729be3ec8410588585b0e89fd7455bf8db810be85dbca
          ded43b9c96bd5641a093a4d6
u[1]    = 00ac232cd219c6fb1c6a814889c6a5a5f7241468f6c7aa92cc6a11
          8e2809677fc3219b9381fd9d2c3cd3d45204f974854b52c88166a4
          e95f67bda3b03a77fe856488
Q0.x    = 00919682cfb480d6e608aecca50465744ec4eda04da80bd2dc766b
          a9efdc13e880e84c8734b4302868b04abd79f0238729d97e7ce678
          2b0a31695d3ff8dfb7b12026
Q0.y    = 0059e1201c05a481e39253745310243782c0d3f686599815f1f585
          1de075e87108debecba6bb90301a7b40c51508202af7eefbbf6bd1
          5c66d8adeb6cbd5c57f7f8aa
Q1.x    = 016406d2f3731e0f0268fa217940fac75738f9885f822625ed98dd
          3a940ad06f97c982a782d5e0b8ba4a197d64448321d0f8bf3e80ff
          0bd66a9bfaf25bb634131e85
Q1.y    = 00ac2f6fb25b10622bc92b9fc09a6febfa13c7d8a48cf3aa29f488
          8ef5a62b8d8430faab51bc268cd943e107692ddb344024c55bf987
          3c8cdc3227740d20971a25b0

msg     = abc
P.x     = 011f919dc22ae35a9f39daafd8a590baae6cac25fdce72a788d5d7
          68b08e76dbcfce09b1098dba43047d620599903b4ded7efc2ab27c
          ff44bd8bf302e515ba303db1
P.y     = 01c2e7c61cd5b3a9ba7298eed4d55f27622fe6775a3fcdaa83545a
          84f31547524a3abc5dcd6a052fff6f58f86f48490d6fcee7a9d87c
          091fd46e34025f8c6734753f
u[0]    = 01eb85e967b18e1bbf6203c2a54023ca26d450e8c58840812d04cb
          4a22334494f7b35ebd5fcdcffe370cee44ec2e00b1cdd27f40c716
          3fa707f7dde13c53cfe57a77
u[1]    = 01d6461a553e91d9a90f2d4305d6e27894d9aee6a10693238a2ad0
          800df97d610a0ca42a211521d8cf723edbe8923aac6f14e0a9e5a5
          4b2304b991d1290d7d238b90
Q0.x    = 010c8f95d5cb19c41c07053a027deb285ac46d6dc0bca6e18bd293
          e0a363e913e0bd670ee3241ba9b7f290f3cc490d8369b3d35dd373
          c3db6c5a9de4d4d83b371a3f
Q0.y    = 01cf7f537359296a1ad2bff6d4ff9369012d9933890bdfa33048a4
          8368000e5fb14f3dad3e497428caf1620c94fcd4617f021af74819
          27cbfce4b4a7343c941a1ba9
Q1.x    = 01affc06abdc12a5034ca64a62e9e49d069cc9166a86834bb0d15d
```

```
          a847e351c6f3fe58a57e9646211ece6babf5b2164c2ab2116e3bc3
          5cdb5d3abb4fc0959a2053a8
Q1.y    = 009f337e3ba7c31dbb54ce4ab15cc77fb02d00de89cde0eb58c71a
          3abe5b1b89c4097454ff695bbef5bc7aae57ba020876050953f51b
          446c25a8c303e682e1f0bd2c

msg     = abcdef0123456789
P.x     = 0191fb5112741260267b1e71ddc7753c2b9b5f568f3d788164d0c5
          c338485e5beb73a496917f3a4eca342d42838fcd884fa464ad8a32
          fddc22afdfeb85759619225e
P.y     = 0168075e06615b595fd1348df72e4081a5e99439cf14fdf9d71b04
          c5a39d7b727d420cc5d26943cf21c8520c96d9b374ca4a45f0a593
          f81e3f9a20c2d632557e7bac
u[0]    = 0001de91de8bc127f45097c096af3917882473574137efbf5e6e6d
          3f77fb55794e2de027ca157f5c337cb6d9214d768804526907b333
          c6ecd9bcd9d5c43297785b7d
u[1]    = 00a488bd10c5e7c0495dab4d2ecfe4a279ef67d2d6bc5d7d353151
          e940cf7c3c21054ae6f98256fc87ef5abf722f4070f957fc5f1574
          9aece872b80429345862ffd4
Q0.x    = 004cc7090abc28807bf2cafd94c8519727b93d4960f8809adf0e13
          b160daf677dba10e56476f0f6defdcfa6490c8015d5bc845fdfa32
          a56a47e0b5535a57ec6f093d
Q0.y    = 001a24cd77ca718008a8f04b317be351cd647fdb7a0ab215a551f4
          c29a2e01f05e4759ca9efb97e5dadbe7d038585b92d31902668ec0
          b5ef436f14fdfe836c65bd73
Q1.x    = 0193195c8a656c5d53cfe8aa6511066a5d7e869406286a1c623f91
          a3c8db5c6ded4c56dd5c4c6307894de4557d36a81954d000e1e7a7
          d8ab0a77bd8923b885849aab
Q1.y    = 015f66f08a2055a42144ba5590dc5bac41d1bf8e999f5ec80b9b52
          d379f81a47a751e3e619fb44d3e230c9e7dd78d1fd3d6e908ad0f9
          b922d2fb603654fef6f968ee

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 0148fcbbe6ddb97429f0bf2f4b5db3919ad5977c6676d6166355af
          f0b0448cedf4ca451ea5dcf2658403df03accb0662d15dbd314b74
          0031e599e11b5e89da9d6dd2
P.y     = 01413b34a200a0e244c601ec27a13087179da06662ed9a1ef29367
          6f127be45f7a10fbf0eb826680dde3ee20638d5ca7357e5478ed4f
          8ad795ce0f014232adb901c4
u[0]    = 002259f4f7c089c393342bb34c767cb4518f9692b1e00d84d6a80b
```

```
          b12dd44694fb43d898a5beaf8ba1a106614b7df0c2e3c750785114
          45227cd310000b7e77bcbfe8
u[1]    = 00788b77dfdd3d6f5dba4af49f93a2bbfcd619be58711a7df538c0
          f57193918cdf0ee483cf25e029257c568784de300bff98c057489b
          df1f9df0b7e99ab8412e716c
Q0.x    = 01cff78630681f055e018f6cdb4dff3723dac546b2fb49340ee8f3
          3a417b790c94b6bd8a5a84cf2dff4ed892de8ca81382540b64b32f
          81c4f0d65053ef9d1b49886c
Q0.y    = 0056ef1568872b438250917cb334dc98d281f0c3265cd7a02fdc40
          8be1baa28cb33f372747597aa2b880c499691bda562e63e413303d
          fb4c9b5df3efd945c9a39b74
Q1.x    = 01b56c7f8f07605935668c1fa6b981870086e384e0bf569da5b68d
          2934402d0365be11be5a97664c0c71f42a5322e1a2aa8b8737a561
          acb161da6c9b9d50381159ef
Q1.y    = 006e24d9bcc739a4e9502fbf3ddf24968c8b0ee15f63ede187056b
          a5a9f8a116ae4c856eb280fb278adbce1678e128a5065e358c1256
          5b6acc6ae4e51981ef97b4da
```

G.3.4. P521_XMD:SHA-512_SVDW_NU_

```
suite   = P521_XMD:SHA-512_SVDW_NU_
dst     = P521_XMD:SHA-512_SVDW_NU_TESTGEN

msg     =
P.x     = 012d13741acd676d6fba94126ad8c1fea0458b1748252642d322b0
          5cd70eed21ec04d82632ad7465b90c567d88c986a504b7971476ee
          e7c57f52c65d1f1380437ed2
P.y     = 019bb624b62227c5e72735a33f0085a12c618670b8df3fa998428e
          1683af28fb77798621a6d6688e7a453cefdc28d710e01dfa05edf6
          6f4fabbcdcd3916a08954ad7
u[0]    = 01d6d1c2ac46a7d9cac9c750ac97c52a079f57d6977ed3e6bfbe19
          fe5c8deba1c89e3ebc54c0313637c3b1b485db6ab973eaca172c15
          2d27739beb4563c37fdb4467
Q.x     = 012d13741acd676d6fba94126ad8c1fea0458b1748252642d322b0
          5cd70eed21ec04d82632ad7465b90c567d88c986a504b7971476ee
          e7c57f52c65d1f1380437ed2
Q.y     = 019bb624b62227c5e72735a33f0085a12c618670b8df3fa998428e
          1683af28fb77798621a6d6688e7a453cefdc28d710e01dfa05edf6
          6f4fabbcdcd3916a08954ad7

msg     = abc
P.x     = 01fdaadfab7c1b1107cfb3aec85807deaffcc52efe635ccacb4ca9
          3241d052c31fa5daf0ec3279d7cb317a088ee8b33efce7481e717d
          b351645b94dbaae92d51aa44
P.y     = 00edeac1ea057c53ba97882162961cf6416600d4a596b667696cc3
          3ae66366dbe2e9c6d91a7daf453e759d3b463c9cb72c93b5b7bee3
          9e16e48b1a83dcf08ea420ec
u[0]    = 01b9501f6589f9ae5f04cd969f2d4f98992790006a15d1171d3b73
          2c39354e1b32ea1eea0a298e2078e0c699302b98a3eea79068294b
          f0d8cb6f93d0d824bfd57bb2
Q.x     = 01fdaadfab7c1b1107cfb3aec85807deaffcc52efe635ccacb4ca9
          3241d052c31fa5daf0ec3279d7cb317a088ee8b33efce7481e717d
          b351645b94dbaae92d51aa44
Q.y     = 00edeac1ea057c53ba97882162961cf6416600d4a596b667696cc3
          3ae66366dbe2e9c6d91a7daf453e759d3b463c9cb72c93b5b7bee3
          9e16e48b1a83dcf08ea420ec

msg     = abcdef0123456789
P.x     = 00c3a8bd2cb22277650e054dc62e549f00aabc316eaeeebc18297b
          2dd1ff41127e80ae9c987a874b163df5583c0fba15215e8680a50d
          0fb9efc4a61936f2d3d83833
P.y     = 0175869ce7bd8df84de35182ae7d61cb58b35ac1f33343a39a69dd
          c21edcdc81b9f3c2ecde52b3ba3d889252611bc8e9c307dfd8cc94
          463a6e556d510aad819b7e13
u[0]    = 018e9e25f4e95c2ee011e76228b20a428c639fbcc8f463e0329ed7
          c424a0cd9619638a5d46f8feeb1e88e2493bb62c9585d3397a4780
          499185944c5502e77f5d6e35
Q.x     = 00c3a8bd2cb22277650e054dc62e549f00aabc316eaeeebc18297b
          2dd1ff41127e80ae9c987a874b163df5583c0fba15215e8680a50d
```

```
          0fb9efc4a61936f2d3d83833
Q.y     = 0175869ce7bd8df84de35182ae7d61cb58b35ac1f33343a39a69dd
          c21edcdc81b9f3c2ecde52b3ba3d889252611bc8e9c307dfd8cc94
          463a6e556d510aad819b7e13

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 00850df274e8755eb6962a34e9c6623c2b7421dac00c6f2109fc2c
          91cd186262b4c979b3deb959f0ce5de6cdc2779cc242f0308896b5
          6f48edebd56244cedecebb56
P.y     = 01ecc97ba7e7638e5e2b0d1c789bb77a8c4478a9f9df7e746fa8b8
          9e5ee27118dd04378b85c3fb1af01a7071ac65cfc028ad98a34688
          4bdf1b7d8a74bc4f3ae68651
u[0]    = 0026d960cc3339da65236557eee01b7e1cfc6bedf7133adef63310
          9c671d6ea7fe773ad258848d77cf7ad6a5f3fc56fc189a7981c115
          e98987c754139901e90123e7
Q.x     = 00850df274e8755eb6962a34e9c6623c2b7421dac00c6f2109fc2c
          91cd186262b4c979b3deb959f0ce5de6cdc2779cc242f0308896b5
          6f48edebd56244cedecebb56
Q.y     = 01ecc97ba7e7638e5e2b0d1c789bb77a8c4478a9f9df7e746fa8b8
          9e5ee27118dd04378b85c3fb1af01a7071ac65cfc028ad98a34688
          4bdf1b7d8a74bc4f3ae68651
```

G.4. curve25519

G.4.1. curve25519_XMD:SHA-256_ELL2_RO_

```
suite   = curve25519_XMD:SHA-256_ELL2_RO_
dst     = curve25519_XMD:SHA-256_ELL2_RO_TESTGEN

msg     =
P.x     = 551e22c9bb52b7c92bf6de89ed0342ed88ebc745b56f41df76d330
          9ace9ecb1d
P.y     = 391fb59b7632b095ac0ba86e5e4bd26d3a0e8a6785507f9e6028f6
          952aadde1c
u[0]    = 029d51ce5a9f1e92ccfb4f9992a833f46680893d8214cc3cc39fbe
          0dc21ec4d3
u[1]    = 34dd938d1a410ae0463069873b482282d0a148bf39c5c5cd00a06c
          deeb053ae0
Q0.x    = 5e97e35ce155f1a9d2d1480b04aa2033fa47de7f193674810da0aa
          7164430a17
Q0.y    = 07dd2f5cfb0096aa5b54c9338c7d5cf51c22ad4161ef97dc4da42f
          c04caf095b
Q1.x    = 64b522bb69567cdf0b1fc77e4c451d655039d1300a76e44bc491e3
          4a652ee2b8
Q1.y    = 1f02d73bad67e92ca65c93e8806cbac16ea852126cd37eb44a1bf5
          c0ca195c46

msg     = abc
P.x     = 5dbca64b6e5b1d52a14834a1d27318c9027cdedd9f2b3f0e043831
          d81df197bc
P.y     = 610cece822c51538d2be6d629f8df3b8ecfafe90794b176bfcda18
          736b3481d9
u[0]    = 41a5a09179d68a50735c5b4aa9268508254a548bbdb43afa6500c5
          1d7904c501
u[1]    = 5626759522ee975d2cfca3d4b26a38fe371c38858900aa404c3cd3
          49127824a2
Q0.x    = 1f5008d8cf3682a81fd9f40243cee0ee557a4d3565141bf34f62c7
          078b58f0c1
Q0.y    = 019bd58a930a486f52fe8c32f463e7753ff666f603676558e8d47d
          df800fca5d
Q1.x    = 64f9ec0d1f4041b7c14d28445d1a21ad80a35cf2e0b422ae2c6be6
          3a1fe80892
Q1.y    = 619cd420135fc2aea1c3026ed7b8cfa3b1ec79255f06e494e4a3eb
          33de56a312

msg     = abcdef0123456789
P.x     = 0dc6ee7e347bc8d6215bfb90f77ddb7135be460271e9a89c4daf6e
          606b90ba98
P.y     = 53218d16891f5dc8706be89c74297c89a611800157fb73fea7d84c
          4771ba9f06
u[0]    = 17e6eb3adb84a1953eec11452a728c3fdcd2e95f8fd6b67fa79361
          d1f09f1db4
u[1]    = 557144a6bb572ede4c57e554eddb2fd126353d6eb50d0ee696259c
          2a1a8e7753
Q0.x    = 56fc43b58f1b64d744c37db72afc2f828842a75f4959e7a988ef26
```

```
          bf9785bde5
Q0.y    = 03eb55be088aa9e9a0a59143d73e4997e61fbf55156f229413b6fd
          3cee712322
Q1.x    = 16ea83da89a4dd5bd2f657269f986856cb78a13de5a10f9cea399d
          317c2cdf68
Q1.y    = 318307e8a361fc56a93cef8097c19e8a601a94a809a31ca8e3b33a
          94021551af

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 3e5bfe6e86c7d42722fef7a41de5e89334652578e88c882a262094
          2d35d9b5cb
P.y     = 14dd0ff480b0b00f3f59eb3c229e9f6583e4eb79a357881532378d
          c5b434bee3
u[0]    = 26e9ad5f584bd6bb708eee63cdd5f3cf9f88d36b129c89f8d54180
          4d2ac0a482
u[1]    = 57bd5385a5d6ea6d49350482801989a452b3476a0081ee2bb5a312
          585ad28eeb
Q0.x    = 5b5f1d868ffcbe4ef6f8a606b3d9767ad30ecad900cfb07277c3ec
          d8593196bb
Q0.y    = 2afcc868cf91b5583dede67b55eed1706dd76a8a184972894a0037
          7360da7874
Q1.x    = 64d8b8301b35997ff212d61899e1a8c953666f4ca4942554ac4eb2
          82eed9c1d7
Q1.y    = 1b2db02fd39064c6d2716f7c775787b5201029f8ab20cd108c8682
          9abd153b87
```

G.4.2. curve25519_XMD:SHA-256_ELL2_NU_

```
suite   = curve25519_XMD:SHA-256_ELL2_NU_
dst     = curve25519_XMD:SHA-256_ELL2_NU_TESTGEN

msg     =
P.x     = 384822128c05f38a5139a72781ee71c90c9b837870746200d9a027
          d5f7fa2c56
P.y     = 3db0b7ac1691801416ba8a76cbc49acfd41f1e31becb97eedfaf7f
          e6bfe56fff
u[0]    = 4ed5a4f4dbfc953d1ade72efc965f1cfa44d57640add47b3b36507
          739ffaf4d9
Q.x     = 1a153f75938296cda751e6894dc1413c8286eec49da74b22be99d5
          efe8708005
Q.y     = 0b0c5538e44ca85142d7be8f4bb5e3f5de6276dc881e026931a849
          146076cfd3

msg     = abc
P.x     = 298d43b25ee3db66136cbd4e182ad53f1e92389ee02f120488d20b
          97acaabec0
P.y     = 288ccd99f3bb33b62dfdd62f17f800f39e9f89877a325c9f873669
          0474f0f900
u[0]    = 4ed68e01a4912df4088e77ae8a22cc47a44943b9ac2e6e23218bdb
          9f67416742
Q.x     = 282ff81ffae65d15ad359823e471d986c3d9e9db12b9ed081e221c
          57de245521
Q.y     = 7c5eca3edc6a55c843bf9d37c7c27305bf09db001f138b8a2cfcba
          1eee06f3ac

msg     = abcdef0123456789
P.x     = 0cec9c2220c2968db5922a249306d3e5671feaa8940a345ba90f29
          5f1a4b7dfd
P.y     = 467885e1292df27c09a5a686566d70010ca91910350d8127c62f9e
          5db1a59dd6
u[0]    = 334a59d8fef7f1f7ec781e5ebcc43f0675ffabd33e472c8156f121
          9063558153
Q.x     = 3f9366f4a010430847777e9af2166fb86cba2747aea03b6a0b1770
          effa107ed4
Q.y     = 495719cf1566779aa98aee1622c4cea88470ad86e90669ed0f0c83
          42ce6151b1

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 5bb54635e3991cf45224a33a71d437a4de23380c6927e1194ff86b
          6773ccecf8
P.y     = 06266c3e11a656d26f32d839f90d43f3e766ceae04e8374d3c2729
          6244ddb61a
u[0]    = 4c7da91907215ccb20b8a064166b03e2cf13abe2393b6729499731
          292f089e1b
Q.x     = 2d10c1e1995ee326f35e38ef2d083a2375cd7181a541db6c7ce582
          f8984867ce
Q.y     = 1f246d036efc124b21ba899e3aa870bac4a7c3c731bcabf5cd3268
          7f6c041b15
```

G.4.3. curve25519_XMD:SHA-512_ELL2_RO_

```
suite   = curve25519_XMD:SHA-512_ELL2_RO_
dst     = curve25519_XMD:SHA-512_ELL2_RO_TESTGEN

msg     =
P.x     = 145f1a4a2cdbb00f721af9fc0720755d082dd3309d585413bdc57b
          79deed66a4
P.y     = 265318a216a4eaed4db4dd4e27abe71b94d558e1b60a0363f8194d
          6cad7502c3
u[0]    = 60dee63d38dba6095eeb95ea343132b6f33d84e809d92a64ba6f34
          8554c94481
u[1]    = 607e4b11d1b17bad8d482b7a4603486b403875740ed3a643fb948c
          95c967bb87
Q0.x    = 2b15e08da01b40a96b46f51cdabf3aeeb75ea9bf223eeb217638d2
          3f6d849a11
Q0.y    = 1aa11a09488c7ee589a232d7b57c3836de0f0b389f2eaf39114e5c
          691ccf6ee5
Q1.x    = 2a263d203c2c54af0205dae921d2f8a24fe3e7aee9f6e401eb385b
          81412aa964
Q1.y    = 5a8ea52fd22c0727ba4a52bd52ff51a6ab9bb70c666e79a1015e03
          9b66654467

msg     = abc
P.x     = 40215373a3b81b38af3f6ced23769f4c075619da8dde18f4d9ae1e
          00f1b848f7
P.y     = 0e84dcc422b1504eacc6d34a178ff8443db269c84d4af738a1ed60
          38074692e2
u[0]    = 7f346b4b77c04efce603d5443c80fae1dc5ac1fb78a859a0aad809
          0fba0841dc
u[1]    = 2741ea7cba3c843ce34e9b1a6cf4981c1be23e91ea3c6f60efcc71
          ba62aae6db
Q0.x    = 44b85219849bf492f790eac07b96b624edb5b266d13a351dae8d97
          8dfb42ce94
Q0.y    = 305d336eaccd259ee67f43861f06039a60bfe22b743cba05de7229
          58a4a16064
Q1.x    = 79a8fc2c4401593e55b05e0f2655b6e07473fcec1c87dfc7c545a9
          5b76f53b2a
Q1.y    = 6834a80ed53e0e2ef8e7ddcc7aab4048975141aa205b26b4ded130
          a376957d01

msg     = abcdef0123456789
P.x     = 593fc49ec025f0485c369962b8e6d62d1ce87cefaef5cbb41b5d66
          4827907900
P.y     = 0b0ff5742d778c4b09c9e8342a1bba2e81d23916228b684db428bf
          8c11ef329c
u[0]    = 724d603fa092b3b2fb26acfc49b69214e8f6b23455e5ad1288eea6
          82acb24c45
u[1]    = 07d99c595eeefe4895921cf06f5fae756f41030ce804c739d712d8
          95bd85ccc5
Q0.x    = 7db3053a52e9e9c5d9f6d9d08500823907e5ef4fd7dffb242053f4
```

```
            d0ed40fbd9
Q0.y     = 0b797789facc33a96479aefeb4e4f91698f43005aada9d71516df5
            ea34431881
Q1.x     = 48ed35ddea926f8b2a55bd20b18c3d22bf737767c7261d1ab3efe4
            123c4cb7a9
Q1.y     = 0099c457a4a9c6ab32d5a89a1f35f6e54d684e58a74fe62442a39c
            2934425955

msg      = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x      = 6cff0ffff35c1bf2f342153245740fbcb008e46640688f9aca30cf
            b763ff9701
P.y      = 6734a600e4f8597d9ae7a27197bdcf5e97999360310ea7df27cd03
            a47e4fb180
u[0]     = 6cebac5b14cec271dea87b00822ce9f4712faa39895826edf9e7d1
            052fdc6a59
u[1]     = 62404e1b9f98555fb42732d16052fa5cfc00a86dd5d5d0a0bed0a2
            56475dcfc9
Q0.x     = 7c31abd1045caba48ea77fbd078bfa4c20ffab70f90a2738ba8ec8
            88dc5bc01b
Q0.y     = 528ed08cbd11986252131d2ac4d3700dbecac75bc4cb66282dafcc
            37a947c7ed
Q1.x     = 2df9dcf04cba473f70794b46a24cc3331b16699e121667171d13ab
            4251a1737e
Q1.y     = 06c4ba3124d13c9f69752387a55977acb5ef410bdf475f8868eb6f
            1f9a15e527
```

G.4.4. curve25519_XMD:SHA-512_ELL2_NU_

```
suite   = curve25519_XMD:SHA-512_ELL2_NU_
dst     = curve25519_XMD:SHA-512_ELL2_NU_TESTGEN

msg     =
P.x     = 3d97d9b446926250ae3112ecb6cb6e610c53a9cecee5b55c1ab35b
          6f07a082dc
P.y     = 07df167ef93c7842c6561b707025d500d8b560139a83eeef03fb31
          ac9ec4c843
u[0]    = 65d61dfa58cf932cd60f65f24739d1afe89442e4a3b2e6ea9d65b0
          865f148901
Q.x     = 1241e151a4da26769778a6c8aa5a219ff608c6c31bea49a0558487
          a102bf6355
Q.y     = 4bf3ec4f91865513d898097fdce2990b6ea8fc5a50de2d9950c50b
          97fc374c79

msg     = abc
P.x     = 17591a1748995feb03b4184f6abf0361dfe9bc64fc461bec862581
          137f146812
P.y     = 70825eeb8f20888be50833e1401511464ee7ddecd6294266177fb7
          95bdcc744e
u[0]    = 1393600fd74775f0d19a8ebd282b05349641a8b079c71030bf1bf3
          70c41acef9
Q.x     = 25c758d9c7a11a43e72d3ddd7976458330fa9e37f63991457fac99
          a70253cfba
Q.y     = 55108cb5971fb8c68e2bd2d869ea52495e8056075857c8656cb580
          3a1c42ee89

msg     = abcdef0123456789
P.x     = 7ccf1faaeb988112434274c401de53e74f14c9afb17c07ce1942a0
          afaa8bd0d5
P.y     = 50df42f5065e8f9ade54e2eec2f04048250c9eeecc9709c58ed09a
          e829cf6d3c
u[0]    = 3e2240ce35b52e6ce7f2cc4c3e57c03bc1619494256e495dadadb1
          0dcc2675e8
Q.x     = 6c70b60bbea45867f1e270af0812137ac64c8931370dae67a41301
          41f9bbacb1
Q.y     = 58e6b785c269fc21d1ba8be61c9c0595b1d5df7574815bba5f883c
          9af79f68be

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 2557e05f806568bacea2439cfb15b34910064a91af6115ebd851e1
          8270f5d1d0
P.y     = 1fc39764b621696e64a4c17a5c2e0051d7b5fba740260a0fe871b6
          fd872f489f
u[0]    = 6719884a782571b55353a8f2946a3e42430f0c769ce85d1a47026d
          491452b773
Q.x     = 64aaf5ae99e65460ba87f7f3c6871a66c3c5d3e671b33fe723603c
          0eed7b9922
Q.y     = 2348578fee0d3c353a6efb0c20f02ee50f69cf50626f3558b99424
          d19bbace17
```

G.5. edwards25519

G.5.1. edwards25519_XMD:SHA-256_ELL2_RO_

```
suite   = edwards25519_XMD:SHA-256_ELL2_RO_
dst     = edwards25519_XMD:SHA-256_ELL2_RO_TESTGEN

msg     =
P.x     = 046f7ac05fd7748eb4cb99043af9e7a7209d8372e27034fd4f6cd2
          58c5557711
P.y     = 0ab78b10ee6f33f339ed388ce1393f80b0b27817b89dbf6ae943a7
          5f93fba38e
u[0]    = 181ac326893428d453e8dc058bc62a11d82c6a8c205749184abee3
          c94ea43a3d
u[1]    = 715cbdfc73cc240655c96e503b6afedbb675de116847afa7c22f97
          71fdf8b70a
Q0.x    = 3903bb757a7e768049f8f0a08b96ad3c034eb694cfae969fb732be
          0996df3caf
Q0.y    = 72008e13e0e5090ca79a85c71e3f82a9d5adba0473e4a9f5853848
          3a00ce9896
Q1.x    = 4378c3e197ad248a484b98711dda3cc4ee3647c88385435f9cd81c
          da1b4909eb
Q1.y    = 726f30bdb55cbd9e00b8801cac092f9f68d7a650829b16de151426
          25b8c2eb0e

msg     = abc
P.x     = 4b0e6009dd4333c3c80d2ec73d123ef69868d3b7547a611eaf7602
          a08627b605
P.y     = 4715be234514062aee8060742fcce446808b81135669aaccbf6aef
          4ffd43e386
u[0]    = 63fce9868bfb9165094ed2b097adafa040a1a00d56720af1e73a41
          c273f49148
u[1]    = 50e9ecb264e462d76203dcb526b00485d1ef568c5ea346d89c6d3f
          7b16a15555
Q0.x    = 0e460620f17e7a3b0258f757d90cde88510dc5bfa415dec91043fd
          b585fe05dc
Q0.y    = 2ea19cc92331ba9d2250d56aaa48f448973b5120734fd4d09048fc
          8f67c97485
Q1.x    = 1501716f21813d93de9eb6e37e5a606236ee0d30c56a46a933527c
          225dd2b05a
Q1.y    = 3b193e355856bb2381aa97aa3e5e25569677f5d97f4983c414075e
          b0bd01f434

msg     = abcdef0123456789
P.x     = 4c937a8324452f3c1339500069d70031bb42124845b599317d6fdf
          46c854d008
P.y     = 045d7dcf705fab83fe638054fdb6af021a67699bd387a59dc3ac57
          cbf798c6a3
u[0]    = 57c9d949302f36433734eb0c1b45fd0b63a018cf1bdda2f04cc13d
          ed7b6bbf4b
u[1]    = 7d224f3cf3cedd36bcb6edb33c28e932d6af67cfa7d84499674990
          e5fb8cade1
Q0.x    = 240dbda5cb37d8f621b22229b02491800cb37f89fe54f14e83d1a2
```

```
                7e55a6c277
Q0.y    = 476ae78a0c17dbb5bd75fa7b16d12f1600bad9fb764329bf50ec17
          8765fdcd5f
Q1.x    = 7e11d53d262007c0db3cee76f485f217cb1e8b7af2250b9a37b66a
          42d58d0937
Q1.y    = 3aac2ba9701325187df28b10a44fd433468af8454d21db384de44f
          4c155bc079

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 1ebcb6a0da0499e59982aa64156f4304bf0583afe597becbb21db5
          6e3ac157bf
P.y     = 6bedadab00ae61939e299f7e5bf00f935d520b5f1f7073e21d89a5
          67715274e1
u[0]    = 2c1413e90118989b8df79484247bf6d480cf242a270ba2ea9c962b
          4ff933c835
u[1]    = 7690959cd6e05c5971a329bee690f5f1f02fca7c783e6f4ed71ca7
          f60aa03a1b
Q0.x    = 0f3209fd7026b4483081e988365ece6919fc41a8502d66e773745a
          5ffd981b74
Q0.y    = 41ea9efc9f94fa2ab628ae99aac817792b3557b40bd8791bedc68d
          7ec85d9d1d
Q1.x    = 2557da4630dc78e33b34d273102fe14daa2d81a60926273724d2b7
          43af4425fc
Q1.y    = 3a4a99bd1b4c83330eed3d02c6e63349f3dd0a36a1260a99ed4c04
          f7992ab48c
```

G.5.2. edwards25519_XMD:SHA-256_ELL2_NU_

```
suite   = edwards25519_XMD:SHA-256_ELL2_NU_
dst     = edwards25519_XMD:SHA-256_ELL2_NU_TESTGEN

msg     =
P.x     = 08226f0a827eb3bb0f0589709b9bd2e6e8107a685d80c6c9cdb40c
          7230fc97be
P.y     = 055d178a04b891dcbeae89f6daf1360a48683a6ee37599fb90d0a0
          7c3d8a25d4
u[0]    = 596f2044f5405f877cf98ae8e5550d449ddd57272f19cc07e8f035
          6e8548d383
Q.x     = 28b50697900178d1a2451b09cc1c573c13da4d9b2afe3be423cd4c
          1bcd39dc2b
Q.y     = 57e58c8840f2e7dbc594ccf45b8663833e34317ae2d956135c5c01
          de25f41379

msg     = abc
P.x     = 1ef41abebcbc9b662c5d09379246412500004048e36edb1088a6104
          3a4e403e25
P.y     = 0ff9bb8cdde0ecf76699e1b5ee3e373e1303e468c736af5517b090
          874e01298c
u[0]    = 22649b4f1cda6266d740d8d546d10fa453af8b8b047945221b4e98
          3ea28f7b37
Q.x     = 7e9989153c21a1fa25580ceebf4793230abf6fd049a4710a20fc4d
          73e61c56b6
Q.y     = 796967a3ab929db72fd3e23395b02bf5391ff6c5a7e7f0abf3a8f9
          a707cb393d

msg     = abcdef0123456789
P.x     = 4e54152535efce2dec85bbe6e314ecbd6b49399954eb2e359706fa
          6af0f9d5ff
P.y     = 4cc40c04402f0959da468ab117680072fdc35d58b21a5faffc7ac3
          e0d650dddb
u[0]    = 56a17c686169e72e6cc2b12785ecf7c423693bd3ca2fd37f901ae8
          6dcdf7bac7
Q.x     = 5a001b1dd41003a45014174f02a77f4ce3017df5959be5b4fe1316
          bed1e3da3e
Q.y     = 2b6a80d085e1637cc20ba86136de04b867cdd0c454627e5f14c5a6
          bdac0cf161

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 363a71aabcfb1eacd545403b3a49ae9a9f46aaaab1c7a2b8eef789
          152acb1d81
P.y     = 6ffd3d900a09c546bd16924210b35dc4179056d5b8c537579aeed1
          4f01454b44
u[0]    = 43f438bbce3938385aba3efa7b4704d0a2896593edc4f99ee4d7d8
          f99e408d9a
Q.x     = 5100fa3465951cc6a7eadafde16a43dfcee775505bdd05ccddb3f0
          9e26102d9d
Q.y     = 4a3846b73f87fa9e46c4aae0bf2634699adcd324d85d0478989aa8
          2a3bbd1d00
```

G.5.3. edwards25519_XMD:SHA-512_ELL2_RO_

```
suite   = edwards25519_XMD:SHA-512_ELL2_RO_
dst     = edwards25519_XMD:SHA-512_ELL2_RO_TESTGEN

msg     =
P.x     = 355199593026fae01f068dcf79adb8dbf2da7b1f040bde080723fe
          aa94b65042
P.y     = 1af40833bed99ac42f445e9494dbcc489561b3995a40e3864a1b1b
          db6ed6ecd0
u[0]    = 5561279bae7b4bffa30d2249dd56904af4e9b5844770979ae0b2c9
          8f8af24082
u[1]    = 05183b3cbbdd97689256838cde5c531baf7a95b1eb9943303706a3
          63883638ea
Q0.x    = 09c5be10d8e2629cc5921040760ef8c8ddd7ffb8bd9cbcc6256f50
          6e870381b9
Q0.y    = 17929f5922ca2c188a2b6c7955c4e4618248c390be69e01b56cfdc
          b80bb8861a
Q1.x    = 04130b6e1f8dc7c32d2eb98d46395fff8579d3be4a91976e2b0614
          ed6f39f8e8
Q1.y    = 65b1c3a24ebe11f4113b6f936f26d6f630de27c21ec0979b08334a
          a3a4349bc8

msg     = abc
P.x     = 124b63773b748a096e505433ee596c120168623ca320910d8298e6
          d90a4a309c
P.y     = 7bdb83b6322c4977ffbb69b18df168b56eec733a0254cac0e85eb7
          90460ee4b2
u[0]    = 3af3dd59fb2b343e836c749290372ade32db6c6aefcfa8460bafe4
          4913d16c17
u[1]    = 0fbd9bb5c9446b308fd4e8d6d37bb40a85ec994fc156d9e2f27c6d
          3cbefbcffc
Q0.x    = 48ec52ada1d034bed2941a3ebaca1bb3fc7018e4cfac62e5cb6a05
          82d85b4e87
Q0.y    = 2769e1a74624974a71d543723bae513305c3c8b2ab8431a07f31fb
          35147e1184
Q1.x    = 23586e9d6a60c06eb7fbf3d66b6dd7c9ed5c8dba065d59b5a71577
          37ebf19415
Q1.y    = 0c4118945ae99cfc6c56ab993488795f30826e7fff95febe3c923d
          c4a523d4ff

msg     = abcdef0123456789
P.x     = 5538ebb02b3351458ae3c37d4e59247be093fd2537f4e18605c341
          96a0b2b3a4
P.y     = 2dd85e9585fba5ab3c22a07cf07b2cf4a19b10dcab49e1a4c8f952
          461e644bbc
u[0]    = 13058c758c796708365f21094f489345f4e30e49b1d380ed064e9b
          7b8d107cad
u[1]    = 319be66d0491de283eeab5f2745e6ab9b19eec8c13e6c8b7069e2e
          6aad42f8ba
Q0.x    = 066b6b75588b6d2532568b3683a7c1b24465be6fa3d06fe61e8bc1
```

```
          42e04afb96
Q0.y    = 222cf85e42efab0c89c6e210d774b26f2d05a4c4262f1bd124e622
          c6fe5c1dc5
Q1.x    = 220a2fcbd88a7fbe61f263ba6b12d37aee8bdfb5a40e6a4fe89ad7
          e438643432
Q1.y    = 1ad7fb8388761fdcb8cc206d87cb5bd1bd6a316e66ce782e7a10ea
          fc5284f412

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 4b2083d31c45aa28a4e5759e73e35d5606be05d9c1090df93f09d0
          b23ceb21c0
P.y     = 160e08388e85b6bcbec8f321ed1de259ffe88dc547a1ce506d61b3
          0fd2378fc5
u[0]    = 547dddae1d6f60997fcf95d741eccc7fe5cf60857585e125dfdf9c
          c6eb987a81
u[1]    = 5dfdd58493b9fab32d3a39e0a9450c82b4308d7fb24ed43f585ce4
          70581a4d93
Q0.x    = 3a34b03ac9cfbccf869c4606664d2bab9f048d306bde0e73dc093c
          5818bc9674
Q0.y    = 3e5593005148b6f4b9a6491610a288191e3264677b03d7e9c8c870
          33053f86d2
Q1.x    = 19031af03055a1efe13c6ef43d88e9ac713bb78cb985cd61985594
          24029f1d02
Q1.y    = 42ece5969a9b335cdfaf2367451ec12c56dc0607f038606222d113
          2d078f0b4f
```

G.5.4. edwards25519_XMD:SHA-512_ELL2_NU_

```
suite   = edwards25519_XMD:SHA-512_ELL2_NU_
dst     = edwards25519_XMD:SHA-512_ELL2_NU_TESTGEN

msg     =
P.x     = 34110392cf2797c527e790379df1a910afa14ae2c368b246a26b9d
          9925ac0c71
P.y     = 115fe9c145cec75332210f75537f89a5af2e9e81928ab63225fcf0
          db4b9f0b41
u[0]    = 50d07fff97e9f2984a1763dab82e1da8bf6628f4bcbd8a5a8b3bc2
          1aac8ea907
Q.x     = 672e3335e81eb42ac9d280a5799f073af446f5855a2214b4b32105
          b01b2a0e23
Q.y     = 69ca23a597a1b0a1f6f90e911af4e6f9457a903622cbb6b9452eb3
          3b3d85a81c

msg     = abc
P.x     = 4a3cd9eca424168031e7c2ef6ea8872919ee1d109e9ca6e2f1dfd1
          9563ff63eb
P.y     = 2a78994752306e4a4eb1c629b3c8625cc26f143f1fe35fc728d295
          222eba7890
u[0]    = 569fc930f70c8b4e50f35c3ae893264757589f11e43aa8bf995603
          1a4fdec18f
Q.x     = 5998e943fda82e321789226093300204ea96af8d183dd7146b775b
          f1d79d1e94
Q.y     = 7467d959e0f62a399197daf58f3d798ce21310d5a1f61e23151e9d
          e278747421

msg     = abcdef0123456789
P.x     = 165abfd4e6ede6db0b8e90e40d4639ed5d7521e4779302c605ff43
          0d0aa9747b
P.y     = 0ca9e2b900bcdfc8b26d705040b6b6a6fd6bfadfd43982a2a59e1d
          069898ae6b
u[0]    = 42bebc0bdffd5ce8f9b875e318a04e451fcfa323e146defc71937e
          a2ed43c839
Q.x     = 3bc0e89d1f93a8c43d16d43ee390da0b60e42184180a4601333404
          296cb9cb8e
Q.y     = 1685bcdbbddd93475f5b0490beebc6a00db1c6ec019c1768ffd705
          246b1ace93

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 0544b2343ce246e25376db1ba4ab61bd91740ce3ca4ec8db31d8be
          d097fb4758
P.y     = 511dee3fcc3a422738c453a58acdeafc08fc76f862e97d678adb26
          f03d0963b5
u[0]    = 6b143c02ab4a45c6a7e71b6a97a71a1fc8a089f7fb83164256518f
          40f15d96d3
Q.x     = 3408ca8aad2b2f343a498ce85bc6e56cccf391bdfbac0b7c178250
          3d3b3b0a1d
Q.y     = 1ca5e123fba67a5f0fb433c3e2423df5518b9ccd74e6e01f652e89
          914f089907
```

G.6. curve448

G.6.1. curve448_XMD:SHA-512_ELL2_RO_

```
suite   = curve448_XMD:SHA-512_ELL2_RO_
dst     = curve448_XMD:SHA-512_ELL2_RO_TESTGEN

msg     =
P.x     = 21f78478bd717f5fdf648512597ff9f3c2a4bf5800d4dd813018fb
          047286af69cd8cfc3b4380c9dc93f3b560b4e63d4bf19c41fcd6ff
          393a
P.y     = 94cd55f3fa7ffd5d1266461a68c5ee03af4aadb77249486c33bae5
          19916b77535239c890552c6f63b9fc4c9f00b067caafe416701b34
          ca3f
u[0]    = ac71575f98e4a4bdb7d0fb683200383a44e26b693fa59ef897a4b3
          27211478fd674def10aa41d0a5c6fd910a11deb010b47ab47b9f1d
          1f7e
u[1]    = 3b1423f8c141e4d8b956eb5fb20b9ae8ed7fe593ebb8ac3edadd84
          e1e54a97fd726950aa9f6d461b2adc037caae51bbcd6ff3dbf3bbb
          d872
Q0.x    = a3193ba63182c9d276e0d8fd9e4b957aecaf830ac14c0652e1db66
          e06f7fb20a7f2b4e5b0ddb0c1212b40b4bfa116cf536c096afb814
          f848
Q0.y    = d6d1e34860f5593191a774e05f5d68c63adb525ff7896a20040c2b
          30a1566d17e9372020cdcf227b98b9985d06a2b9ba04e88f715f89
          9702
Q1.x    = 8389a1b3131219141579dbc52d63d913937e1a0e8493f82fc46141
          ce355a311ea2b3c5bd733aa087aa6482fe5e89049d1e356460ed94
          871b
Q1.y    = 21214a80f9a4fab7ef59bfebadb7fbdfcc852c1cf518239ad7c175
          c1a9a25f8740fa9f1300a2de04c848c60e44e08fd25b8f285ceb81
          895a

msg     = abc
P.x     = d0ffa9b8551e7295f3e663c993db473e7a049bdb0c0c7b23a34297
          5229d9958c5371db48a25ea758d0a1b8d5cab5c1fffce003f036da
          ba04
P.y     = ddcbbf8edc1509c2a39ec2426b81e495a5ce33d5dcd96357625f27
          279f0366bbf287061e984666a142d280edc1329554c353db819843
          fb4f
u[0]    = ed9f4bb9bc0253536c45d34688b3096725db693f5af1659ba48ca0
          67aab8ca495594d9d94d3cd8bd394b79c566c57baa16cb330db3ac
          e7ed
u[1]    = 2fca5d5201399fab882009d46078874a17337843a86a64863ef931
          6a3ef293395c4de2b306dab0002169295fe2dcbd973a8210c6b63a
          8236
Q0.x    = 3244dbebbe141c71e59bee53507f3dbb5e2d7ea35cd6767eb9fe18
          b2c2901c717cd578baa564db42de9b227643bf0b13654a0cb95041
          6bb9
Q0.y    = 6aad2e7445bb01f43797bf6b24efe796f6925b162f97b29addab33
          24fb6b5556158a73c9b2c32463ffc1de536cd65d1ca49fa064621a
          e503
Q1.x    = 9f8608ff76d7f998ad3376f1dfb7a3441bec0ec6d4f9950dcd16eb
```

```
              84894a271e547781249f59c25c82535eb07fb103f4088eac45aa01
              5bff
Q1.y    = 511a509c545ecfe9999e388fdededee610ae572affa42e6f587cef
              07b5626e5c02f4949fcd959d224e31d825d131de32b1b510b7d551
              eac0

msg     = abcdef0123456789
P.x     = 1dc040d287f05f0ac0769405216d285d17b3ebaf9e9554c8ffbc7f
              25de178b528fae889b7d90e59f8becd5bf395414b05adbb4354018
              baf7
P.y     = e9730fdd89aa2ed9be03aaac27fc6f655856055d8ded93ecc63e31
              8e21b240d592e60781b207539f04a28a36b2f7511229675bb38a6e
              de21
u[0]    = f070b8ef55cdf617fd63f9e25a45529ed60b40dffea4ff42ca90d0
              e96a2591cd01fb9dfd5b3a1bdf8385d08146dc4343cbae367a2e41
              677f
u[1]    = b55db04e2d841a72ba1194eca52de0635083520cb9956430b24b39
              e4de5e412272103039b52872281c789eef41bc45eed4af277c97ef
              30d4
Q0.x    = daa6c0bf01ee5ed465450c0cacb515acb022301cef25ef3e324dc0
              288a746830f1da8b68b93e2b1db6ed7c9444d8235fc9fd4c5cd86b
              8bdc
Q0.y    = b158d1caf233ce737998d56f34e4b652c9574d0a35fa3a2cf4f4e6
              c3ad261ac54eea2cca895a9ffaf0397b760a7e21a636f1b201d91d
              e5b3
Q1.x    = 8087fb4658454128559baeb6b2a87e20ef66b308847ea76a9340bf
              c5f5194654ec626b0d12b88877b5b2848968ba5978ffeb6d7b0ada
              c3c9
Q1.y    = 8c5109d75eeae5f3e15faceb1d5852e72d6a8287e72bcb9759c912
              d19f281ed0e7111e0ca72720d46f9b314b67f2025a2e82b86aa715
              4104

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = ea5bec7c3f36f844cdbdf215067785d7677776bb7d19675d1c3912
              4ecfa828d5ee7eac1c580630be0483b3740003012f32cdd5f6a18d
              af4f
P.y     = cfe11f4cb36327bfbd1c711b53b839eed8a71947c1209439608523
              3f815703afe136c85a93ee527308b30270c4085147a2e652dfa102
              f742
u[0]    = b63a92b6bdd618be51a73c6bd34af23975c73dbe9962be502c08a7
```

```
          563feb81710a798034c70d9ed7bbe25d81b1c8c27c2f850c258447
          8cc6
u[1]    = cb86f19ef00c35982a9c104c8b82cc8d2a199bc31f25bc879b51d8
          8168fcd8ab305f4331573ea12a6bf69f7ea76c0f1a84be987dcd06
          6f50
Q0.x    = f50b3633869f3c82e88b54790df222e706f611d2ac6f7f5a2b3e7c
          f20e3cae24fa247da0b326f04cb0513c842d6ccc479f84355896c7
          d678
Q0.y    = 306e0233225931cddaa4fa8ee5aba40ab06852590226a7c3a7684f
          43e12304791110a64751f27ad8a17b93ce94977ffdae8db2ca72fc
          847e
Q1.x    = 33b99bec6b0311d6433d57b762c732259e46ff134f59d16e6e9c62
          d77df8dd597c41493976d3bf07ea19d82487b3f84979e53cde2b8c
          c833
Q1.y    = ef6ed1bada9b7261695ca14282c5956d8e4df3c9a3d6bd7606e15d
          130b9278325809abb3c8a4e30918128a66d2a9af777fe3a0ba07aa
          fdd8
```

G.6.2. curve448_XMD:SHA-512_ELL2_NU_

```
suite   = curve448_XMD:SHA-512_ELL2_NU_
dst     = curve448_XMD:SHA-512_ELL2_NU_TESTGEN

msg     =
P.x     = 2183d97000fd3ee7e910b7cfe21d9996168d4f4c8b0a711e66170b
          d1829a35eeeeb9200932e7ca4cc7faddc1b716c4808a4adc9a68a8
          59c5
P.y     = edd9a4f7154b646f21dfd0387a7696066eac44ab1cf0da578cfff5
          5e60da581c19d0436312b989141b764a5ab641579443aadda96105
          33b6
u[0]    = df946392828b16784ba146354c781b3c25a28b5ece5b9c8088b684
          0f871d7578fe4850276ec01f2035197f5882c1e6506dd0fd7338fc
          2ee5
Q.x     = db4e79e5d711e6c82d9aec2c3d8ab7f41148a7fe1fb6d7ff586214
          ee9f752b3f331627a2f258f289c5fe38536e8e213a9ea06695c051
          0a38
Q.y     = 491f677655e9bcefcda66b92cdb02d41b73604ef7dc23a69af993c
          eca5841ff33312b9b844a02f3054056ab2811da3ff1fc34251b6d5
          f80b

msg     = abc
P.x     = 88cb9634c61bbbffff09d423857aef50b15b42bc4389d6c66e179b
          de024cbe75d5cadcf7be4156f1aa2a40f709d3f3b02245d4a67176
          2c80
P.y     = b6565ca3d04f50d8d22511af542a55118f8cac4c6235ff42e0c3ca
          066c2f98d5038022fe42009bf5435e8c049027ce3cb9592784f3ec
          4449
u[0]    = 801a7cdf6b9926e3617df4617178487dbb92a7ef310170316664a6
          4f878a20214af859bf691e1e50499dd937e09f5e7b040c22a521ca
          2466
Q.x     = a1e939400b8336cb04ff098cb2801a340d5200a85f5f56befb7acf
          f92c5a58041e4329fc733f52c0d0e68e3686331e8dac6fef3e88f6
          17ec
Q.y     = 0a4d08b25bcd3d73ee88eccc7c97570e9f824b769df692ed22872e
          847707ded4373defed7a54a798d3bb05a61c6de281b05a3cf12a2d
          ad80

msg     = abcdef0123456789
P.x     = 9b4d581f3b10bb090d5fb78a4dd65f386c6530b97c0f0432e092d5
          80f554fe432d66463b3b4a1a660845a66b9ecf98ae5316df6ffa4b
          fc2d
P.y     = bcd1555142aab27c4cc26e1c5117eec0b8cb858e4500c1b76d7717
          3d5c937b384f1705e9efa049f259ff7d6b8adbb3e9ce8f9b8a0e7b
          9944
u[0]    = 7309e98e2d59af9c84deb4f8c6210df9ec569d724ed7ecb22d56b1
          c9f64cb49b04d905336e0ef3f1e204553a0adc136881fb3d31dbe8
          2d0e
Q.x     = 2918222446810f1de7b0729776e4d63ac4e48e2a2279b53355cccb
          9b20251bb4a311938fa3ea0ffe3c046e68f7c896a2b39e79a9f157
```

```
              3094
Q.y         = 8d18c4b0458c7085c592272ed20afa34998f7486025cfe326c7afe
              56411812579ff94778aac3017b28b2d08f8ce164b1923edd813ca5
              7f6a

msg         = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x         = 5e8679afb70fe836ab83577840234764b9879ddd9509bd3fa6894f
              111bc21b18718a3b2fb4faab519cd3096bbb9c9ffe828c8702f770
              0c95
P.y         = 97d37ef305860f65ea247da63e85a5b90f363482ff38b5e051bc82
              759d1a9c9bd5ddc3a812ea4453390ea08d2dd3289f10cbf3df194c
              b895
u[0]        = 33921076a158e463b80644c2bbcd14d9f064c96e8d0a451c357d6d
              2ff361e33f9eb0b4dd89b30b1a7b79a756cdd611b4c2d1e416c621
              cec1
Q.x         = a9757d017a87fc07ed43268cacac61c3bdad1e1b73c7fa8c816fca
              1343b05d051eeb66650a114422a1554905fafbc866cbb8f5ae13e1
              5c18
Q.y         = 1fdc90b44483cc5a5c90e83954366456a2762938b7e88b3c8dd09a
              e95a58d2c78a4889a742f3cd8ef1a03e88fed8e3c9647fd9bd8af8
              5267
```

G.7. edwards448

G.7.1. edwards448_XMD:SHA-512_ELL2_RO_

```
suite   = edwards448_XMD:SHA-512_ELL2_RO_
dst     = edwards448_XMD:SHA-512_ELL2_RO_TESTGEN

msg     =
P.x     = e54af3058170d7b465db0cf48ce21e28659aa04a5b490f868b65f3
          8c95d83164a19b47ee029c2b64e1e5a446dab02bab4be161c11e18
          2ec0
P.y     = 2e38750489aa0bd7dedb33212525be1a04bfcd97223c73fae32dbe
          f05c2041db24dd9b62b0af44dfe44948f8fed4f38c991124073066
          23c5
u[0]    = b4d9d1895cb65553ddac80fb1fdfa6718943967975fd3227be7884
          f30794e6c7c41ec24b07e13c3822bab3e7514159429a721f2b6a46
          6d0f
u[1]    = e235635e9ca91fd4e0384606e2bf1918d84afd759cfea41bf56397
          2e1b82c2d700194ea823852e4ce83815b33bc522e13d8c361ac311
          9f39
Q0.x    = 2c665a5226d6cd7983a08c910651f624c904dd8f1009648e645d04
          e381fb04c718b14f1041092e46e3935401a3b85eaaa7a0fbc0ce67
          9980
Q0.y    = f15d91cb50d1d9818521684b761aaaf0fb1db0278c718cce5d8aad
          7166f42d79d0f1210e7957b170daae9be3e353f9e3465a5bbf93dd
          5c64
Q1.x    = a36c98ec2b442335d5debc9d1a59c34e6a7e67d02dddd41256cee4
          e2f705401778f02a7cf21b85635f1eb9f34aac0985575d26b02122
          beaa
Q1.y    = adf6a6544cd208538e16de28e725dc674152c20a9311fb9df60894
          17ed33f2dcd85176759845b957d35f32b9c56b1396796e33439b4c
          6a54

msg     = abc
P.x     = f583e90dcba4b48f7b0a510bcf5e3b1ef4a73e4280ab8836480335
          177cf251299ef33fed2c7492082f5a443297dd79dd4bdf7f30b1bb
          bedb
P.y     = 241ec020befbf4fe92528713ef2275445547abd94a7f71beec8c8d
          edf42f68d51ecbd6dcf5789a317c85cc2daca79f4ceddc8d5e9d14
          9d84
u[0]    = 18ea51720224486cad1aecd722b5748a4e6297df7603f264983ce1
          519a471097d328e7871c694b57e83de0cfec01629f9b1c36a66478
          a949
u[1]    = 0c9de3b0e748d03e3c914ddecca5d4507a11860ed73abc2c953ed6
          2811fca4fe3f154e8da9520c019d9c8389eff12936e09c2e0f8e8b
          300a
Q0.x    = f2a84e3d13ab1107dbd4db40555570dc1c61479a9d5aeee22941ab
          a0feae6ad7f018bae02e5086b772e5b15f070f1cb1eb303e6108de
          4951
Q0.y    = e3d62ab209de4e0b23f20d9aa84b9b18e9c6500211106c1c043eb9
          38647dd08c39babb90b7a3928fb55741744ef15791cc563111412f
          46ad
Q1.x    = 27f62515c82d3f6fe0edf0762d3220b3c3bf4e8d466b6524aa2cee
```

```
          13aecfb7f041ed2247f5a78283293f971bee806231f77851dd2650
          cbf9
Q1.y    = 1ed1a0fba069f682816bc44e1cb5d0413bea5ea43d6c8de600521e
          fcc0e56cbdb745d1fd8b53c1d88d26eb68e09c404e5805067d3bf8
          2e64

msg     = abcdef0123456789
P.x     = d03b676a730b4a8418793154bf4e6a6d8bd94df6964429754346c8
          2275929ec8448feece8b929a053b4513ee90d1d613509aeb81d32c
          e42a
P.y     = 41e6d33f904844d51c59aa0e97423d8060821da3f40a10f9572487
          a3917c5ef1d2cb67f43d6bae65670a1a54233cc927fbdb97a77afc
          1161
u[0]    = 8be92c1e97e9bdfca76029d928899510db3e2a84a80dcac58383e7
          c0bf0776721bc9b99204a68448a7effa87bd63172023dc63ea71e1
          2e8f
u[1]    = 61801a7ec7847b88a512df4c9f12f04782db61c37ab3e1fa21b1cf
          255e2102b34ec3992d79919b6df7831043d28c5e673d46c34ae65a
          43ba
Q0.x    = 159754771764f81704aff2e7849498787f03616c01fb1a232904cb
          b720b68adb06643ec0445314314355c29fd191a7b1447de4a1f3c4
          4685
Q0.y    = befc0bb9d5697a31ec8611197e6c5c0a8299ce93d8b17705b05aa3
          a340cf7efc0379955fc908d86c031f5c475cf7a172d8cfb81c9869
          764c
Q1.x    = 8c8ca5ae4be432d0d27d0eecd4acde82c555a4be4b297c6255b954
          4af4567c03ed07eb1f71f119f5083dfe69773e72c873ddfe1f0cf0
          3cbf
Q1.y    = 6b8ad4dcb2ff061a9b6f94da5e13c6443e0bb596f22d1a9320f06b
          094a6a4253f02d224c05717bdebb477c33537dc3d1c1a2a840596a
          ba73

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = b2ec3e837f23e4dbc82b80a493d202a24bd55c4c304a6f88dc2c1e
          976ccc3d385070829916c923232f845237d60dfbb5e23760197ffc
          9177
P.y     = 5c0312f8c7578b03f816468d079af61aa36e6c6473886308a2d0e7
          d31c6506de9bf8a489dcadd1a2ff86a0160cf199f581074d604820
          3f0e
u[0]    = c561c853c080a370e155141b5c902164324b31b2cdb143a8732170
```

```
          cbacec701febc8fbf986b8821c341c14304cd6c5aaab59b33a157e
          8564
u[1]    = 1540013e32bf3f64c7380adb160b27a3b7fe7f2f53d19f3ee475ab
          7d21088cabbcaffc706a6fb47e4c8ff55fda04e325b7c21c0e2b96
          0323
Q0.x    = 7246a708d161231deb74eed793dc0e7c71edf94d40c9f9f74d23de
          ac9cd5f450c0b4ecc342c7777c2a23b7f7fa526f000341916e3646
          f781
Q0.y    = 12a84e91cf8b70ace832411c2803455e08dce856144667422ae49b
          8f80341f2bb965db7ace67911b3b97ba5f3584ba08dbeb481b2116
          d32d
Q1.x    = 01090cb9f4cf9a02dcd065ed65195b6957e1cd9d7e0e46870c2515
          13d151799dcd2fd3ddd70543df850d7356742b7976b44455ed53ca
          81a3
Q1.y    = 49715e3ce7f5f9200f527c0c703119023df3463c242d114f6b0ce0
          f10a45bb3661c87d5d8cedc137383170df7ab63541bd6d04bbb33c
          62c1
```

G.7.2. edwards448_XMD:SHA-512_ELL2_NU_

```
suite   = edwards448_XMD:SHA-512_ELL2_NU_
dst     = edwards448_XMD:SHA-512_ELL2_NU_TESTGEN

msg     =
P.x     = 47fb0126fc722d16fc3d7bfc3b0b801571f058290408ecae2bf8c8
          cc35e0fd702d0a169f6d9cb0cbe24ad226ae78aaf2f77dd906806d
          3de3
P.y     = 4f8aec8c3c24ab883ad2f8a7803f74f72a0ec20daefd63c96e89ff
          eac238436b27316569ca3885fd3a694e9d7337b3335564ae27db78
          5bb4
u[0]    = 3b8a62f7162ce426ec4562db3e5aa5263b1ef51caba6a81f562623
          ba219b9449ec128eb5f8aba2115ae738d548308689f42fc33c03c0
          dc16
Q.x     = 2f37119d6f0854731f013563fc174fcb93b0099adfed722cb988e7
          6a597b05d489ae3f559485b0400997369a851f0385673633cb8811
          360f
Q.y     = 7ca637e249804a7cbe2c07f339e6996a29c612a4b95fa3462335ef
          3f87b44f8a700c33839e1abdb455abd6434c01b24ccee7b62179ec
          b165

msg     = abc
P.x     = 9bd168d0151f8487c63f956e33614ef45ffb7f501d13241e747101
          26a681d5d860396cca691c813c6a95fb35f3c4265a76f3a24be96e
          680a
P.y     = bfa4b2acec2af0523016d12dc37c7d0695127b1a9a7d281b3ee63a
          8b105e58a8f98145d701d10bc47b0fa3344385323658c00bee6b17
          855c
u[0]    = 784cee1a9ee364242b453257e00d8f6254063ce4272ea59362d6bc
          dc8c5c94a5c7eb49107188146f06c8ee886b1598d1aa148deb8cb4
          cd65
Q.x     = aa75440eff5a95296a5c1304d00fa7b1d93c5c81ba0b2088d6a2aa
          2c978466e8cf99ee64bfe20f102710c0ad85d55a8df432e020975f
          9dc3
Q.y     = b0aea1a5cb12e9a2fc5a67b36e7adb7a3b6b9a4438fde9b5aa8a93
          6edbe940333182504fa4439d945c263aa774f60ca9b1f0f166373c
          b4cb

msg     = abcdef0123456789
P.x     = a175d110f5aff27311798c4ac46d7021bf6dfa4a30d7d9a9086e43
          60aa6249d87991a2a4d134eee1b1fd81393404146e34de1966a44c
          d6c9
P.y     = b6cae3d5c4d7876952344335b02504dbb31a73af59fd8288af3ec8
          2a7b69f87ca99c9c6846098a4ac1c19030cdcdc97ffed7c3cf9524
          2724
u[0]    = 9daf8a9fa44d817c75349b5bd94a76ccbe8438bb0c41f7544a207e
          1c0e3277df27615a0ed4dfaa7a754c7d1bb12634f974a84b4eda16
          b1f7
Q.x     = ee1c91e69d29c4b66b121b60d53910ff44ad8c30c61ace9be71f76
          48ebdcf781fd2734ffaca664fd2e63af563a18a30c37a056e31c68
```

```
            bb0e
Q.y       = b89afa69e9e60b820a4bfa8d519101ec63c5429bb309caa961452a
            56697ad0c268bb04265d474317e327fa926d4f7ea99b3a73546154
            04e0

msg       = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x       = bd4fe82690a62897671e2cbdf5f9dee5b42e0d33e7fc300233a594
            a1e0cd296ad9f001962a0988d68815dd919c85236061f7945d7f94
            0fa9
P.y       = d776f6a3c24ef5e937ee7e889bcb112abd51d644a299bb44ea8965
            bc13675136ad429952a1af5833c4968c7322dfcaae2a19d2cd124b
            a4d9
u[0]      = a3b2b2f3d9feb982c871dd951354ee037d40da82c221afcf201f80
            3d2af78a0c8e5869146d5cc4c4831c3199153ccaff2b0208b4f969
            98c8
Q.x       = 10231374b2bf13eee3d6bd6c1f27e0ddcef4dbdbe06133357170a9
            4aea79f95fd86dacda7c48e6e6d326d940c09fb13759ad18abed8b
            2d0f
Q.y       = c29bb3c5305c70c482ab2ef87a8e81b0335f7d84982f57615661c8
            ee4c9c260c2804093f1315739f9fd907aaa98afc92092e6f86ef4c
            a173
```

G.8. secp256k1

G.8.1. secp256k1_XMD:SHA-256_SSWU_RO_

```
suite   = secp256k1_XMD:SHA-256_SSWU_RO_
dst     = secp256k1_XMD:SHA-256_SSWU_RO_TESTGEN

msg     =
P.x     = 733fddf3612d3516dc9ec8b61759c242df573668f43bac67b0cc2b
          b8ebda7b72
P.y     = bb633f8a914f1dd1da988b49cf11250c2fe66a396f6c8bf72981d9
          e140eca7d8
u[0]    = 3ab7d05ca97119b5f91a5c9a780d32d19e6a4ed12c78c7544fdf3e
          7db31e045b
u[1]    = 87ca26bfcd0173e4511d86805db971b22cc15a2b95a2c63417ea6f
          dfa37ec043
Q0.x    = 68f789f85587d190e04957967b5a2024b43b88a745ffb6c9f4206d
          1506f17dbd
Q0.y    = e22d2020b3b618b8997a1a1f49c092f4d519dcb49da74c9516d892
          fd939533b5
Q1.x    = 5201629b2ad52b0ca37aee01a1e13b7e2ecc91ce3c279a6667293f
          c1afe06c5c
Q1.y    = d8ad74bd3f93081816e48cf21a5d5f3a8bae7dd426c901384f88a9
          f5cc0bf1a9

msg     = abc
P.x     = 0b71db187b0a619331e953035a46e6d6f9f141b9f8b109fbe38bdd
          0b8675a2f8
P.y     = 84dea7a654229b50d487c558bd67f056d9e30b35ecb136a2de4d62
          bff9903e66
u[0]    = b2567533bc95dbc88e637fbcb9a748a8786708194d31c8df97f79d
          171838272b
u[1]    = 99bb9aa261e47e4bea6e7c8d0f37ff62ab8ad9ad8ef0bb11784a97
          28446f9069
Q0.x    = 3c894e4fdf3cba59f70e3f9c900d3900e9a0526ad278c5fb007c62
          65ba023eab
Q0.y    = 26c9899ab4b83c02a879bfe028bb3eedfcca598cd9aada9d064b94
          16a5b98607
Q1.x    = 27ceb73d6348acd61fa1ea85d2d7bc45eb533ea41605e4fb306ba6
          17fc8885d3
Q1.y    = a115a75b37ae1fd23ad8652dfb37fdf5133b6a47cccb0a2c211b70
          317bc93a47

msg     = abcdef0123456789
P.x     = 7d2bf71041e0267a226b3ff85180c577b637d63888274b8f01ca6f
          f210432f51
P.y     = 62ae2567997680b685c9af79d64e572765f579e380cae90b3d706c
          c67ed7b9b9
u[0]    = c574c75aba2addf84986d0c27df6871d24fd982c779e350bbb7da5
          e3dd2fc110
u[1]    = 2b48512e02509982099eac4d319bc4b6b87c405fc2578437e5e933
          c977d101f9
Q0.x    = 8d3b7d21f3bd15f5b31d57556d956c103b942dbc94516653922c3f
```

```
             e8169b26f1
Q0.y       = 28c2a16424a0d1cb735c9f144c4357731afd6490ddb8047254a87c
             34dba67699
Q1.x       = d6f0424cdedecc6188910e7bc1c0851b7d9e9284053ed912aac2a5
             a59392b2a6
Q1.y       = a4568d0292dc767a473ab390b53fa8db7d7ea97ea7d90793c74bd7
             c6e391de19

msg        = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x        = ab878eaae0e759a225f472c09a201c29be494f648a25f6ec1e1196
             fc87258749
P.y        = 5e4942b57b11bbae42d4a889019d1a54813905e5178c44d7cdbe46
             59ba5c8d1b
u[0]       = 389de71cda24753bbc3811e2c5d54195063deb1254fcf448179b82
             bb55b9e044
u[1]       = 934f2e766b69d419308ec93a709c16470344aaf9101440bc05e074
             ed6cbe3e3d
Q0.x       = c96b220ac2d34570de856340a6146b4942f54424ec1bc1e05b63d0
             c756066e9a
Q0.y       = b204aaeeb0fe9139e8c8dd64cbbe37208449a5613e081e12c2709e
             685899f200
Q1.x       = 24695b3a6d95538cc99a1ab11af2ebcd14c8f676ae66b396922c85
             9528a6e9e6
Q1.y       = 2f20daceb860082bf39bc17831e90ced55416d54737b0af30a8db7
             8b88c9ee52
```

G.8.2. secp256k1_XMD:SHA-256_SSWU_NU_

```
suite   = secp256k1_XMD:SHA-256_SSWU_NU_
dst     = secp256k1_XMD:SHA-256_SSWU_NU_TESTGEN

msg     =
P.x     = fbf556f8dd33ec00571f567ab0b366389f8e58e57daf9c0ffee251
          3dbe3b12b5
P.y     = 8483a69cacd420ba70f49a28a6ea05964db89477954ccab7a58931
          43aa84daa8
u[0]    = 59deac5afa2bfedf03f0275173a38da0af4779da257375a22f98ef
          06404e5f6a
Q.x     = fbf556f8dd33ec00571f567ab0b366389f8e58e57daf9c0ffee251
          3dbe3b12b5
Q.y     = 8483a69cacd420ba70f49a28a6ea05964db89477954ccab7a58931
          43aa84daa8

msg     = abc
P.x     = 77858441c354c88ccb02c0743d081ae80df8ccaba688cb0bd3b0a9
          02dfb55077
P.y     = e23759ff36fb09d5ca6c148d178058b48be637253d275c937323cb
          fcd08a1a1d
u[0]    = 8008e395ec68247cd1fea7cccb6ba6aabd0f58aae8f3937dc297b3
          e82bef9a4a
Q.x     = 77858441c354c88ccb02c0743d081ae80df8ccaba688cb0bd3b0a9
          02dfb55077
Q.y     = e23759ff36fb09d5ca6c148d178058b48be637253d275c937323cb
          fcd08a1a1d

msg     = abcdef0123456789
P.x     = a505dac19323f3ec71aa84a5984146923c59b0720634e7667d675e
          506994bb52
P.y     = 24c8166d66e7fb737696f37e818ce8656e7d32acf6d24ecddefe0e
          d15dde620d
u[0]    = 7b846ac5db51b662ea619efea1fac238a5efbb1c124b2c2e39d112
          a62d01d267
Q.x     = a505dac19323f3ec71aa84a5984146923c59b0720634e7667d675e
          506994bb52
Q.y     = 24c8166d66e7fb737696f37e818ce8656e7d32acf6d24ecddefe0e
          d15dde620d

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 3e96c858026157c6f05132e4784aea7dd9d6c0d92f601f068a1528
          3bb42252f1
P.y     = b110972547240d89f1bfa9af84bb1069876f8c96291395bfb0b573
          4e8339cc13
u[0]    = 83c53b55e8e12d06d2c82f94a3eb57349869f2736a7a340a8b8731
          089031b05f
Q.x     = 3e96c858026157c6f05132e4784aea7dd9d6c0d92f601f068a1528
          3bb42252f1
Q.y     = b110972547240d89f1bfa9af84bb1069876f8c96291395bfb0b573
          4e8339cc13
```

G.8.3. secp256k1_XMD:SHA-256_SVDW_RO_

```
suite   = secp256k1_XMD:SHA-256_SVDW_RO_
dst     = secp256k1_XMD:SHA-256_SVDW_RO_TESTGEN

msg     =
P.x     = dbc32a97a6da03a615b70cd6e0d58f2653c03394513634e977db70
          a9195345a8
P.y     = adb868f0f9be71c8765063883adb65733635a206434b8114d98786
          ce049ed9ec
u[0]    = 24e247f642985dda504f391474a6e9d6cbf9d809e11e2757f8ecb9
          825361e126
u[1]    = b1029a73ea8550005f09b4ce0f61cf5dc270122bc371c3143f355f
          6d18338656
Q0.x    = 31d9cec3b1cd1f46c0e777f61b4974296f2f78644bf03d6c723cc0
          75a371b608
Q0.y    = 8b50c0b855f66f608c01bb24d3c4d14998e7c95c73cd6da727c752
          01eb4258ee
Q1.x    = 35536a1bfee29f3aa387c69cbff2077c33b391c3dc9ec63a4e129b
          ef5f771721
Q1.y    = 7a1c8cceed0041fc609c3b4b3c2dfacb311bd08848a6fc8997cde7
          97bfd04ffc

msg     = abc
P.x     = df5fdf0c1ab9ad2519e04b17f3c31fc521209f95302d6ea3f48e32
          eba3aa2fc1
P.y     = d253f780074bcd75b013920fad69d47643fbd1755cae12b4c160ae
          adb23cd34f
u[0]    = aeb022d4b701a2fc0adc5a8996c9e81581744a2f7c286f92560d0c
          8b5680608f
u[1]    = ee2c23054e45fe1893f0bc6006e2adb6e8b7feac3c9cd9bb08093b
          1ed6ab2b9f
Q0.x    = 009642c108e1f574a6d48f171bf2bc856c7a60a61738fd201c2fd5
          3ca474fbcf
Q0.y    = 694f5dee86fa93e1cf01a0f234599b3d68fc4305d27497417cf71d
          3df5a122b5
Q1.x    = 583dbdbb235ae3a744df5308c79bb4a50208e7e3ae4b9e671a4060
          deaa52615f
Q1.y    = 80b6dd1edda6883fc8340801b8bbbe8d658da185bf6c08b559ea69
          bdaf23a7c1

msg     = abcdef0123456789
P.x     = 4b99c1e00552b18f407fd5906d3deafd2e6fa4e16e07eb27728207
          a028467f3d
P.y     = 0dddf058cb1f6e0bb11cb0be1d5fbce9fb40c89fe75a11df7fa53a
          6028d3bb4f
u[0]    = a40b91d1f83bd226fdeac8bfe0fe6253b5e1f4f50558ae91a6a162
          e83aeeec46
u[1]    = f8a893f876e825e887b9d1d0e637f96d77538c107507c333684420
          36e11f5df5
Q0.x    = 2bd23602845f0a4877ad2252c51d3194dc9d50934431cc6d0ee4db
```

```
           63f68ef056
Q0.y     = ba81d93499085d585c1f7a7466f2a47b526ae55bcadd9cfffb4739
           cc9c5ab22a
Q1.x     = 7f3d94fb8fa566daf060a92d6a25113782d9b33bc538f1bab8c7ae
           88d46618d7
Q1.y     = 22665f0e9b8c2fd7d7c3313776dd03d6354603cfe89107d13e1b52
           5b79aa55c9

msg      = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x      = 90ca97c050ed2603d5d366c8129e9fda737232567f60229e453dc9
           ffc30893d3
P.y      = 2a796c4045a9e665ca79cea40d8255d05e59073c8446eda8b59282
           1f1994dc9a
u[0]     = d13ba9f2012fa3b19fc1aeaa35b7eefed4389909491a05e2c88dc8
           e81f847d08
u[1]     = e99cbb84be75005216494eb95979e7567fdf9c257d8305e146597b
           056195a5d4
Q0.x     = 010aa437bb197d03387fb57756f9fda7e9e73f3a8b38bba8a11965
           6ba178bdc7
Q0.y     = ac2ba13a58e302a3e0cbde5fa2503b5eba5a30a6e774555b87d92e
           385fa52746
Q1.x     = 2fd8097bb4131beaebcde17658abba2f4135041b7bb5a9e5d699fd
           65f836928a
Q1.y     = 265f01d0f5907d426ea805acc38eb22847c0800ad6d2cc62b6d6c2
           dc922a7bc6
```

G.8.4. secp256k1_XMD:SHA-256_SVDW_NU_

```
suite   = secp256k1_XMD:SHA-256_SVDW_NU_
dst     = secp256k1_XMD:SHA-256_SVDW_NU_TESTGEN

msg     =
P.x     = 51da4962212a8025a5a849706970cfbfe471f681c049405086a7df
          5a2db652be
P.y     = 649a8ba766c9ccb8232d665a731c59bcd06a918801fc106ebfc889
          01a4ca2733
u[0]    = 84798acb6f9c692d909d77598f179d47205d69cca9aefa36f87ff6
          7c1aeae0e7
Q.x     = 51da4962212a8025a5a849706970cfbfe471f681c049405086a7df
          5a2db652be
Q.y     = 649a8ba766c9ccb8232d665a731c59bcd06a918801fc106ebfc889
          01a4ca2733

msg     = abc
P.x     = 7617367b4d893c4f7a0286cde9c5b92d828244a3d19fb8c17cecd2
          5cd08ff439
P.y     = d129b568725160c2af879746d2a9087a07f94e526ed3ee5b2b1d5d
          bb1cc0bd33
u[0]    = f794d108d20381eee8a178c90e75068e232ca8ff0f391c21153016
          226ac56eb5
Q.x     = 7617367b4d893c4f7a0286cde9c5b92d828244a3d19fb8c17cecd2
          5cd08ff439
Q.y     = d129b568725160c2af879746d2a9087a07f94e526ed3ee5b2b1d5d
          bb1cc0bd33

msg     = abcdef0123456789
P.x     = 27e97147da349e94eff05e11f816d97eae95bfb4b37d1656266219
          99780cfea4
P.y     = 9fdb96f46863ae060cc05b1b36478e28d97c1e33ac743715c2d36c
          7feece408d
u[0]    = 11db2699d5241142e8afd619ea3ba174033b0b5fbc15d197fd7759
          7237b7851b
Q.x     = 27e97147da349e94eff05e11f816d97eae95bfb4b37d1656266219
          99780cfea4
Q.y     = 9fdb96f46863ae060cc05b1b36478e28d97c1e33ac743715c2d36c
          7feece408d

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 443288595246459c9f71d0c2e4a8ec9b9bb6f002e0c5d9cc853ca8
          7c6d662701
P.y     = 82b74662cfd559e77c915848dcd4ca77e75b80802c6bfba2fd4aba
          9ff929b5aa
u[0]    = dd6da75f3df111c0c103e4c238a387d997cf6b11b435f8ae88340c
          f3c3729fb6
Q.x     = 443288595246459c9f71d0c2e4a8ec9b9bb6f002e0c5d9cc853ca8
          7c6d662701
Q.y     = 82b74662cfd559e77c915848dcd4ca77e75b80802c6bfba2fd4aba
          9ff929b5aa
```

G.9. BLS12-381 G1

G.9.1. BLS12381G1_XMD:SHA-256_SSWU_RO_

```
suite   = BLS12381G1_XMD:SHA-256_SSWU_RO_
dst     = BLS12381G1_XMD:SHA-256_SSWU_RO_TESTGEN

msg     =
P.x     = 14738daf70f5142df038c9e3be76f5d71b0db6613e5ef55cfe8e43
          e27f840dc75de97092da617376a9f598e7a0920c47
P.y     = 12645b7cb071943631d062b22ca61a8a3df2a8bdac4e6fcd2c1864
          3ef37a98beacf770ce28cb01c8abf5ed63d1a19b53
u[0]    = 14700e34d15178550475044b044b4e41ca8d52a655c34f8afea856
          d21d499f48c9370d2bae4ae8351305493e48d36ab5
u[1]    = 17e2da57f6fd3f11dba6119db4cd26b03e63e67b4e42db678d9c41
          fdfcaff00ba336d8563abcd9da6c17d2e1784ee858
Q0.x    = 02f2686965a4dd27ccb11119f2e131aefee818744a414d23ecef4d
          b1407991fdf058f0affaee18fd586a9ab81060ae20
Q0.y    = 0341a16c88a39b3d111b36b7cf885b7147b1d54b9201faaba5b47d
          7839bcf433cc35bb1f7b8e55aa9382a52fe4d84370
Q1.x    = 1357bddd2bc6c8e752f3cf498ffe29ae87d8ff933701ae76f82d28
          39b0d9aee5229d4fff54dfb8223be0d88fa4485863
Q1.y    = 09ba0ec3c78cf1e65330721f777b529aef27539642c39be11f4591
          06b890ec5eb4a21c5d94885603e822cfa765170857

msg     = abc
P.x     = 01fea27a940188120178dfceec87dca78b745b6e73757be21c54d6
          cee6f07e3d5a465cf425c9d34dccfa95acffa86bf2
P.y     = 18def9271f5fd253380c764a6818e8b6524c3d35864fcf963d8503
          1225d62bf8cd0abeb326c3c62fec56f6100fa04367
u[0]    = 10c84aa245c74ee20579a27e63199be5d19cdfb5e44c6b58776593
          1605d7790a1df6e1433f78bcddb4edb8553374f75e
u[1]    = 0f73433dcc2b5f9905c49d905bd62e1a1529b057c77194e56d1968
          60d9d645167e0430aec9d3c70de31dd046fcab4a20
Q0.x    = 119cc1d21e3e494d388a8718fe9f8ec6d8ff134486ce5c1f971297
          97616c4b8125f0dc568c59836cbf064496136438bc
Q0.y    = 19e6c998825ee57b82c4808e4df477680f0f254c9edce228104422
          494a4e5d40d11ee676f6b861b6c49cf7de9d777aef
Q1.x    = 0d1783f40bd83461b921c3fcd0e9ba326ef75272b122cf44338f00
          60d7179995a38ea9c66f3ce800e2f693d2634a4524
Q1.y    = 017b2566d55fa7ee43844f1fa068cb0a11d5889c11607d939da046
          697c8ba25cf71054c2a8eb2189d3680485a39f5bdd

msg     = abcdef0123456789
P.x     = 0bdbca067fc4458a1206ecf3e235b400449c5693dd99e99a9793da
          076cb65e1b796bc279c892ae1c320c3783e25062d2
P.y     = 12ca3f12b93b0028390a4ef4fa7083cb23f66ca42423e6e5398762
          0e1d57c23a0ad6a14db1f709d0494c7d5122e0632f
u[0]    = 11503eb4a558d0d2c5fc7cdddb51ba715c33577cf1a7f2f21a7eee
          6d2a570332bbbe53ae3392c9f8d8f6c172ae484692
u[1]    = 0efd59b8d98be7c491dfdb9d2a669e32e9bb348f8a64dbf7e47708
          dd5d40f484b1439109a3f96230bf63af72b908c43d
Q0.x    = 1614d05720a39379fb89469883f90ae3e50995def9e17f8f8566a3
```

```
          f6cfb4fe88267eac1dc7834406fc597965065ef100
Q0.y    = 1060e5aab331ac4940693a936ea80029bb2c4a3945add7ae35bce8
          05e767af827c4a9ffcb5842fbc50ab234716d895f6
Q1.x    = 0f612cda21cee750b1ccff361a4ce047e70d9a9e152e96a60aa29b
          5d8a5dcd25f7c5bd71bb56bd34e6a8af7532afaa4f
Q1.y    = 1878f926302468949ef290b4fee621d1172e072eda1b42e366df68
          fc87f53c35583dbc043009e0b38a04a9b1ff617efe

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 0a81ca09b6a8c05712396801e6432a87b14ab1f764fa519e9f5158
          16607283fe2a653a191fc1c8fee89cd30195e7a8e1
P.y     = 11c7f1b59bb552692288da6557d1b5c72a448101faf56dd4125d84
          22af1425c4ddeecfbd5200525064657a79bdd0c3ed
u[0]    = 134dc7f817cc08c5a3128892385ff6e9dd55f5e39d9a2d74ac7405
          8d5dfc025d507806ab5d9254bd2334defbb477400d
u[1]    = 0eeaf2c6f4c1ca5cc039d99cb94234f67e65968f36d9dd77e95da5
          5dadd085b50fbb11489167ded9157e5aac0d99d5be
Q0.x    = 0a817078e7f30f08e94a25c2a1947160db1fe52042626660b8252c
          d339e678a1fecc0e6da60390a203532bd089a426b6
Q0.y    = 097bd5d6ae3f5b5d0ba5e4099485caa2c505a1d900e4525af10254
          b3927ae0c82611be944ff8fdc6b278aab9e17ee27c
Q1.x    = 1098f203da72c58dca61ffd52a3de82603d3154c527df51c2efe62
          98ea0eeaa065d57ba3a809b5e32d9d56dade119006
Q1.y    = 0bcbd9df3505f049476f060c1d1c958fe8b34e426fd7e75424c9e2
          27d9c4d3edbd5eddb8b1e89cc91b4a7bd3275d4d70
```

G.9.2. BLS12381G1_XMD:SHA-256_SSWU_NU_

```
suite   = BLS12381G1_XMD:SHA-256_SSWU_NU_
dst     = BLS12381G1_XMD:SHA-256_SSWU_NU_TESTGEN

msg     =
P.x     = 115281bd55a4103f31c8b12000d98149598b72e5da14e953277def
          263a24bc2e9fd8fa151df73ea3800f9c8cbb9b245c
P.y     = 0796506faf9edbf1957ba8d667a079cab0d3a37e302e5132bd2566
          5b66b26ea8556a0cfb92d6ae2c4890df0029b455ce
u[0]    = 0ccb6bda9b602ab82aae21c0291623e2f639648a6ada1c76d8ffb6
          64130fd18d98a2cc6160624148827a9726678e7cd4
Q.x     = 0dddf77f320e7848a457358ab8d3b84cbaf19307be26b91a10c211
          651691cd736b1f59d77aed3954f857f108d6966f5b
Q.y     = 0450ab32020649f22a2fca166a1d8a59d4c93f1eb078a4bedd6c48
          027b9933507a2a8ae4d915305f58ede781283325a9

msg     = abc
P.x     = 04a7a63d24439ade3cd16eaab22583c95b061136bd5013cf109d92
          983f902c31f49c95cbeb97222577e571e97a68a32e
P.y     = 09a8aa8d6e4b409bbe9a6976c016688269024d6e9d378ed25e8b49
          86194511f479228fa011ec88b8f4c57a621fc12187
u[0]    = 08accd9a1bd4b75bb2e9f014ac354a198cbf607f0061d00a6286f5
          544cf4f9ecc1439e3194f570cbbc7b96d1a754f231
Q.x     = 12897a9a513b12303a7f0f3a3cc7c838d16847a31507980945312b
          ede915848159bd390b16b8e378b398e31a385d9180
Q.y     = 1372530cc0811d70071e50640281aa8aaf96ee09c01281ccfead92
          296cb9dacf5054aa51dbea730e46239e709042a15d

msg     = abcdef0123456789
P.x     = 05c59faaf88187f51cd9cc6c20ca47ac66cc38d99af88aef2e82d7
          f35104168916f200a79562e64bc843f83cdc8a4675
P.y     = 0b10472100a4aaa665f35f044b14a234b8f74990fa029e3dd06aa6
          0b232fd9c232564ceead8cdb72a8a0320fc1071845
u[0]    = 0a359cf072db3a39acf22f086d825fcf49d0daf241d98902342380
          fc5130b44e55de8f684f300bc11c44dee526413363
Q.x     = 08459bd42a955d6e247fce6c81eda0ad9645f9e666d141a71f0afa
          3fbc509b2c58550fe077d073cc752493400399fddd
Q.y     = 169d35a8c6bb915ae910f4c6cde359622746b0c8b2b241b411d0e9
          2ef991d3e6a7b0fafabb93c1de2e3997d6e362ce8a

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 10147709f8d4f6f2fa6f957f6c6533e3bf9069c01be721f9421d88
          e0f02d8c617d048c6f8b13b81309d1ef6b56eeddc7
P.y     = 1048977c38688f1a3acf48ae319216cb1509b6a29bd1e7f3b2e476
          088a280e8c97d4a4c147f0203c7b3acb3caa566ae8
u[0]    = 181d09392c52f7740d5eaae52123c1dfa4808343261d8bdbaf19e7
          773e5cdfd989165cd9ecc795500e5da2437dde2093
Q.x     = 08c937d529c01ab2398b85b0bff6da465ed6265d4944dbbef7d383
          eea40157927082739c7b5417027d2225c6cb9d5ef0
Q.y     = 059047d83b5ea1ff7f0665b406acede27f233d3414055cbff25b37
          614b679f08fd6d807b5956edec6abad36c5321d99e
```

G.9.3. BLS12381G1_XMD:SHA-256_SVDW_RO_

```
suite   = BLS12381G1_XMD:SHA-256_SVDW_RO_
dst     = BLS12381G1_XMD:SHA-256_SVDW_RO_TESTGEN

msg     =
P.x     = 045f87745ff759f9197e131ad83d47d635dc36a3e0c7e4a1be5e1e
          ffe5e63ac69c8f34e6c3aef9c5cf28224922788367
P.y     = 06125886a03f883740a078313d5fa6e4a68b9c0394eb75f77c65fc
          8b44db3f4ef933ac6adf341bc45fabc7907afcb832
u[0]    = 04a74117c448f7aad70bd41328b3856de638c0e10c9ff344295a04
          b90db0f2afe80ff2da62e091793d6e52bc70b28d47
u[1]    = 06f44093874c190fff1e893c847a59601d5d5ac0aebd85ba36b390
          6a93173e0d2fcf6bed8013ee2679ca337f21f8f053
Q0.x    = 0d8deaf1a9fe87c5710466d0cd554b243a041a97c15228f1c65be6
          244b5e1a575f4ab2762a1cc9ff18d45d4f1494e2c6
Q0.y    = 090dce51930eff6bccfc7cc2920f180c398b0a97085ff9e4841367
          e701a0f28616bef537203d27b0656f3b7ee52ee0ac
Q1.x    = 0a36d0d4051eb32d253b069d1d15c5b69cefbb75bf38d66d43b374
          32b34f88b2553bf063c6533ca87112c3e95c295ebf
Q1.y    = 0bdc425e77f44ac13587b17dab2b1b9d3e3501be1fe1c56c7f3701
          fbe53b43a37263c04fd9aa1b7f58b1b63fe6cb29fd

msg     = abc
P.x     = 009a357691a6f7b2917d9a34ba64d896d40b49733fcb3207f8c146
          e20fffb47823198a26b6ceeb01215fc3422908020e
P.y     = 03fe44c894c107a8547826b60f577b90f80c63f899ef9dcff94daa
          dae180ad803609337c9ec97d6d9b8ba306df7a9849
u[0]    = 145191560d1db38062a0a2e29469d71eb035f888cd4bb2792ddf88
          ad63c93320738a3e4c6e199bb286e26efaad665a0f
u[1]    = 17f7bbcbcf64dedb58af6560e2b2d08a678e4a57a0b042978840ab
          b77f25b33090035675ac30bb461a8955a9d1ebb411
Q0.x    = 01b50bac3377c4e764fda5fcbe9bb477c3becb8cf18a3026a88ab2
          a0fbf6d8164d60d85ac9e67b6712fbb26b10ec4597
Q0.y    = 11acedae54a20e1f8d58fd074e1b7de957203f56365b52a09824ff
          9f8d191356914818bdb35faffe8657ad24e1f5db5f
Q1.x    = 16bb1fce160d3fb5194b275a907cb03bfbe8713e830f2a573acc26
          782c65c9a6e88d96c6489c96dc6f65815cb5ab2662
Q1.y    = 159720d02fef12d1a40b1c91336908d72b0eb1fe54a16a914c5ae9
          710a8fa8dfad035ac6c39708c88318fe26113f28d5

msg     = abcdef0123456789
P.x     = 04eb09680fe48598533932907810fb7681e60b3689cb138454bec6
          27490c5089b6dd755556e52a36c3817e98b62d7497
P.y     = 1763dd8bf6823d9a22124d22a4ab3d93d8a9603ec80b4a40905b26
          664b16033fa6e73a6155c9bc4c6faa42bf911ffba1
u[0]    = 06ed3e25d860fc574e482bbc0a09c5f44216ad44d75ea499cf905e
          fe4bc3c5fd3e94df483d17501395f325d3c8ea7925
u[1]    = 1426c02b7de4bedb4a301c7d2d4a270f672a8fc448471f2c3dfcf4
          3846b102e592ad6e055c2d5149fcbc21f5fb6002c0
Q0.x    = 164e16e858fa1e1a4ea52152ffe0a42f1e6ae1258dd830eb7aa664
```

```
          34d7d5689da945e29c61766ae213bc1878363a3d4c
Q0.y    = 077cc4c8f74f33a0e6036d32c4407b3021b0be82e0eba80a1fc119
          f69a071c52f25764dbd72f2cafb420264c079b7691
Q1.x    = 06a856de4eb5651a21c68c7e85796c16e417d7d97cb3e1b74bbc40
          42b340e647fc41cc570bd160fe9917d32a47bc8b5e
Q1.y    = 15c4e89949bbdb253d333018a1417150f8a78187c343f5bbdf25d9
          62c513df1d96ee1059facab028549ebe7716164466

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 0915842b42c2c4d3b509823c60c1fad834784ff451855f43390b80
          c3b6985d76aadc6ecfbb4b42a07921410d6821f0bb
P.y     = 052873ee0b444dd8337ce403636d680cff1e9402b7a1ce2ab210bf
          f11a83fe4e14216fe96efe3f344c1a2ec0fc1b2c5f
u[0]    = 0da5000e88db641d86b6aa2711b88b9bcf45fc95689dec569bd9ae
          6aac6b19e96620920f3fb806d0e72a111889983c11
u[1]    = 0367d0cf2689a9e972f01d9d325a3e1bdcee67af7f39fb71030869
          457ea93d0039e58bb92cd058e45ced781def78ac79
Q0.x    = 15901778217bbe602fe603edfa0518009197445f34dd2184450fb2
          a1a5088945c32d31f6a9fcd9e3c4333557a11c18a3
Q0.y    = 126fc2597e155f4aeeacb7a039b5a0012be2cc1433d8c097b6c8e9
          e6be9b5f279c0decb019a6995345e607a36681f8a3
Q1.x    = 0dcb7e280b34f7caf24928ecb126a3bacbfeb4b70612d60bc7cd36
          467ad58cf33925e9c78635cdb7a024c0a3031846c1
Q1.y    = 066bbc62980e8bf7664f2b84ee69f3ed2c1a49ef1f8c0d35d709ef
          1ed149f79cc93a65ccde5194d333710496f087d41d
```

G.9.4. BLS12381G1_XMD:SHA-256_SVDW_NU_

```
suite   = BLS12381G1_XMD:SHA-256_SVDW_NU_
dst     = BLS12381G1_XMD:SHA-256_SVDW_NU_TESTGEN

msg     =
P.x     = 126690faf7d2cc7838d18f58ae94e9eab1d042fb93ed7245aee5a1
          a3ecac3f1935df62b807f9bd7be1018492b4ccd087
P.y     = 0d2fe772107e935ddfb916da6dd69cb3c9b112cbd7452a3da77e16
          620f58c06db9c1eb19ab031995c96dc2d0c16b5031
u[0]    = 1754372e423dd665efc2703d235a5b42ad5609ee9872c303a0b900
          b152d5ae8d64c2d67764a6f188b87d61445cf5634c
Q.x     = 10c4561604827f4687f7bf5a365411577c6c0298910d6a67685c67
          5f1dd0b59ac5fbb46b729b3389499aaaf5b11bfc06
Q.y     = 114963eba7ffc5e3bc4c86a1dbf73b4952bfd60fa94449b2adc81d
          ee0e974e4afd228a1af4fb1d4ede14cdc727deed0c

msg     = abc
P.x     = 03929609010843a52432b815cb5fb90d6a5ee8a10cc75c70d6f1bd
          64642a0057d2e69187ea3fff0a34e7e72b1cef5948
P.y     = 196b4734ee0b1845619e8b871c0a2ea556660dab119868ad5542dc
          5b782b0df61cb6d2a1b2cd174a80cbc9f8cde6d470
u[0]    = 0a55c5bc65256c488bb068e3b1d5d2fc4acb3017da1f7d9a3a9036
          eb5880123b0aad219aedb4836eab28e7a85d171054
Q.x     = 0f5b63a00bd63757fab41dbc9b47b679650d903a83934eb920a130
          97d58dd5f82068c059bf1de52971373c3cc83c427e
Q.y     = 03f2547ce0337e25a9fad27335f499b054cc1ab3e1c29366959599
          b5d362602bedcce56c0e039c4ec0fbbdf041564892

msg     = abcdef0123456789
P.x     = 107485fecc3be4b92f8cc35f1782c67742d055c379e6bcb7783639
          6edebe728aec4ecad0290bd1bbd5be7047ccb8554c
P.y     = 0603d69e02982c8a0edc6fd84848fa9d9c75a0d7a372eb1ea26722
          45ee2d9ed672cd1a564fec3cb77758b966316d5c09
u[0]    = 06983540e75c0e7a36976a66cbfee71e05c9ea2cbdb3b0f25aa932
          9e02ee95895fcde7262dae46c370c148e1027c4819
Q.x     = 09c7ef760508a7d4a226869cae2c556a8773ebf5d4bbba33b0cffb
          bba598105bf7549cd33d52ded62be72f718ab0ac9c
Q.y     = 0aa4882a7806cac7bb3bd6404000066b7c6ff14dc590efa26f2740
          b1af2f3be2b50e1d5b99b37d723d9d3b0cf089bf62

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
P.x     = 06c214d69a4ecd4fd0230f2dd2a8988daff27c597acee1ec5e6045
          e0848894d1435f5c143f4226fbc27a2045921c2499
P.y     = 179e0869d8c73643e0bc27ebe3178ee297ad366c9a4f416a0b0ac2
          265ff763fbd8c0bf37e637d081190352f4c1961f74
u[0]    = 011a4d12d373940b946029aef4bbdbb880810b25370d1f5b171e2d
          68091fabc9a4c8fc4f39c9b0ef98c34b5f90070a9d
Q.x     = 0598fd95bcd0a790e6921a51f212f5194781521cf32fcea581be2a
          5cbfd2a8fda8828141a24168689a273d258b1fe083
Q.y     = 04d0dbee266de33d9bee538d188e209476ebe3b34e58a19d0ea6ae
          542219b3f3549685ef3eef82991feb23ddde098285
```

G.10. BLS12-381 G2

G.10.1. BLS12381G2_XMD:SHA-256_SSWU_RO_

```
suite   = BLS12381G2_XMD:SHA-256_SSWU_RO_
dst     = BLS12381G2_XMD:SHA-256_SSWU_RO_TESTGEN

msg     =
P.x     = 0d3b02ee071b12d1e79138c3900ca3da7b8021ac462fe6ed68080d
          c9a5f1c5de46b7fe171e8b3e4e7537e7746757aeca
    + I * 0d4733459fead6a1f30e5f92df08ecfd0db9bcd0f3e2f2de0f00c8
          f45e081420aa4392eade61eade57d7a68474672fc1
P.y     = 09cc6f7b3074f0c82510e65d8fc58f6033e03ba7358005a13e2bbd
          7f429b080f29731ef08c3780c9e3c746578b96b05c
    + I * 0011531b8e08900a4f6f612e1e27432961419ce6a5ee3ec904a535
          88982d36ec4ea37be80b6cb7d986b38faec67dbe44
u[0]    = 1921dd796efec0b5f2ad9037e73b7470e6e7c85e6b7bf6e6827729
          442cd01aa55ac765ae451f497873400e90a814105c
    + I * 0838662da53724510bb7d677b7caedbf3c4e19e55b42c9872c2af0
          96272b26cb53cc522413cf4c54c1f7691bcdabfc4d
u[1]    = 12bcf2aa2ec29a9dc0b77c942869af340047a27b57f549ecdd2e93
          fc5e63d0364fba13941725cbbac1d1e8c6e8605d5d
    + I * 04e892eca8729b9a2b88b4b0b33e8c8d4ed52f7d0fc860e41f09de
          0a05c48f605a7a9d603b60db4f9ee49f888e4f3273
Q0.x    = 06828063f239b9607a9668fc0a59e5391c1bf24ba42ab5b4801126
          66e3835c45f29401bea70d6f529d07e33bc017c1a4
    + I * 04545e6b5318ec1c10f75791bc97142722370b851b7829987ada82
          8de5a9b6353cb2c4f8540bf0821328c983e7eb887e
Q0.y    = 06fc755b65fd277f7e1e4a6174dc11fedddf1baaefc07d04c3d120
          e15ca2ec61f489711c4af949bf6a5d26a885db05e3
    + I * 068d9e1e428ed301e425e633603ea9ce5ff02df898aca1957fe5be
          1faf2ff42b64485cab41370a50c1e7e9742c0a2c6d
Q1.x    = 16af5b519e92362ce94c0ef76051510266c5ace22d2cf9c3c1ef20
          856e4669523df27c30e513444b436cef2082983814
    + I * 03cb00c0a0370bd301cd45b0c08e6feea3adf07052e83a3f2ced57
          665d7484f5540db310f649387701731ff185b460da
Q1.y    = 025310a94143c55175eb4929df2d8294b0268f3011850bdf46bdfb
          82749233e8dbb8070d623e3fb8397b079504d6c5aa
    + I * 087d9d2c1cd911f8cd9844e60011a1d1c02d61a831bfd33b58c436
          d4305935abd3207c553c0145fd3ae1bf18c82011ce

msg     = abc
P.x     = 0b6d276d0bfbddde617a9ab4c175b07c9c4aecad2cdd6cc9ca541b
          61334a69c58680ef5692bbad03d2f572838df32b66
    + I * 139e9d78ff6d9d163f979d14a64c5e57f82f1ef7e42ece338b571a
          9e92c0666f0f6bf1a5fc21e2d32bcb6432eab7037c
P.y     = 022f9ee5d596d06c5f2f735c3c5f743978f79fd57bf7d4291e2212
          27f490d3f276066de9f9edc89c57e048ef4cf0ef72
    + I * 14dd23517516a80d1d840e34f51dfb76946c7670fca0f36ad8ec9b
          de4ea82dfae119a21b076519bcc1c00152989a4d45
u[0]    = 0b7b2d371fc970671ddf7bc9ca4a70a1bd286af4487b497e460c0b
          44d405d73db576f8a08d59416cc976d4b1d0100775
    + I * 0e86d0eb2d34c34fe8b2a1f2d999fa3dabcd504fdb4beb57e79756
```

```
          b08fd75b0a82660abc6026ecc4ccf327a522587b38
u[1]    = 10376d048c060df1c5017a363144c482892fe2ce0061094327b8bb
          e49a713ce795726aa23b5402a271e9f1e7b9b6c7ba
    + I * 0117f2ea63015e192d759f11a658a002e06112147d90f00d742972
          2456b9a1c63fef2dbe8df13168e3bd40af2fb959f3
Q0.x    = 198d5f5fad8e1594ed98ee3d5f5b24b58e6cb8ae37372f6028e7be
          ffa0e7a16b0958e13f92f322f513b85eacdd88d0c2
    + I * 09288d195828e46e7c058b22115af5c1bde20cb7462a3f9d648818
          8fa937e2c0bee3aa188622312f87e12b48660a66d2
Q0.y    = 16a0916a0c42492649701f0520c075d4cc66d74cd8fb4f9c6d2631
          cac1bb48e357bae0b97bb7f87f1b08539ac944f46a
    + I * 0dd21353972061024a92db26e51cd97246c89a884ecad67a1ff8dd
          1da73a8397f54d41533ea2ac48ee4f5817cb09dd1a
Q1.x    = 17a29041f55a2b23b3a601b8d4f5a3e85b75cdc52650f26470884b
          8b367b20ef8f7e6f0ee8969b994f4132344a68f023
    + I * 115d814e4f7973e7a960df53b494c2a0a1cd7a42ce6650bd7f46e5
          5a2622dbf6528392ab9c8ff45d3fd97bd7b1dcfe67
Q1.y    = 15749a8af1d35fcb409ffc3336670a8b47a117f81670394183c316
          d7bcd2a49b2253ced38b7d00763a2fe4afb51f0336
    + I * 10624b1845ccd669e5edd403d68a603fb43e382c6570953dd9b9ad
          d472e422d1098eb45380fa870e14f6927252c4e667

msg     = abcdef0123456789
P.x     = 0ded52c30aace28d3e9cc5c1b47861ae4dd4e9cd17622e0f5b9d58
          4af0397cd0e3bae80d4ee2d9d4b18c390f63154dfd
    + I * 046701a03f361a0b8392ca387585f7ee6534dcec9450a035e39dc3
          7387d5ca079b9557447f7d9cad0bd9671cb65ada02
P.y     = 07a5cf56c5ea1d69ad59c0e80cc16c0c1b27f02840b396eb0ea320
          f70e87f705c6fa70cfeb9719b14badbb058bec5a4c
    + I * 0674d1f7c9e8e84d8d7a07b40231257571c43160fd566e8d24459d
          17ca52f6068e1b63aaae5359d8869d4abc66de66b6
u[0]    = 0022182b07cb11d26cbdab43e0d696297a7dfe1b8dd2fa8ded11f8
          58bf25ab000adb1ec319cbfa42d1107a3ec9528b33
    + I * 01160e11ac26a46322b4867a0d66cbb1d8b8f78e88a3771b7a832d
          18c65d65297692e9faa1f65719c9ea621578003c37
u[1]    = 185e096fa6e05479e1f3ae4148fd4de985c73e414f9a9202d3930d
          59a09d90d87e545522a91a0d24c6aa3e2363a48a41
    + I * 08e234820b6cdd9229490f5c1e05e82b8fe7b1efab9dfaabe3ea41
          58f0f8da855daf1e1f5382246187d317ccee520a0e
Q0.x    = 1729bbdeef9e902ab2e2bf6f90e3800231397ecc36b0b53d33ecb1
          73bd682ef45a51e691d7c884965fb530cc85d6476d
    + I * 07fd016eb7f3785362f75a0150d9e73d5ae13631c491075d73eab5
          c3b6ceb8391d909926d0c519fb83fbe889dae667eb
Q0.y    = 15ee2194b053071cd1d40bacbb2650b5608d22d12ddeeae9fb1192
          1e475ffea6d1c008fc390f231aa14589365c6937c7
    + I * 15345785b7ba1db6cf6ec9f652dede47c86b6837b2c43f3a9e6984
          f95feecffb84bb5963df655068a0ad6b8d8a762fc6
Q1.x    = 18505ec8bfa125df7ea130e702eaa33a89961dd24ad06b3b3452da
          15f2394d0abec06aa3b4e9433c32fa8a7c6ef874ec
```

```
    + I * 17c0d91f4c363a7ff183deeb4308fa5e8d61c0263b9d0ddcf304b2
          758e2b556695fe20636b4b7a2cd4909c145a81c884
Q1.y    = 140b4d6603a96ef9de2a71a8ceec992aa72eb8c4f08d28de11310d
          cfd4d13dbb68734001417d0c1587b9082b593ab9ca
    + I * 0ac81f1093f8be742b331c1c04e9cb0fb75ac72e87ae5da9fa395b
          043fb83fbdabe9e54331ec3a3a754f845939b118f6

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 0161130ef4aa2f60f751e6b3dd48ac6e994d2d2613897c5dd26945
          bc72f33cc2977e1255c3f2dc0f1440d15a71c29b40
    + I * 06db1818f132a61f5fe86d315faa8de4653049ac9cf7fbbc6d9987
          e5864d82a0156259d56192109bafddd5c30b9f01f5
P.y     = 00f7fab0fedc978b974a38a1755244727b8a4eb31073653fa94959
          4645ad181880d20ff0c91c4375b7e451fe803c9847
    + I * 0964d550ee8752b6db99555ffcd442b4185267f31e3d57435ea738
          96a7a9fe952bd67f90fd75f4413212ac9640a7672c
u[0]    = 0034f33d3e0b2bb1e396fa3716a02682ebfefe6b99e986c356c725
          f4bb787714f66fca8aac0581b538ae255aa69aa8b2
    + I * 169a32a329b295e56423a29a2fa15b259f5e27f992c1391b3d333a
          4a050d8264cf146b1baa641e609ec748d74d6bfcd5
u[1]    = 04028afd52de566f85dec8fd409112d34f09ed3b617b31bb23b0a9
          6d76080d1dce671a910785cf63d4efcc20112f4a67
    + I * 06d00ecaf61b0f972b521b223aeed36d1e4e1e6308b36dea9eeeb9
          17619499d06615c275ea39cc4d7db697e4b697d40e
Q0.x    = 08420c5b8d9f73ddac45b6ce050a8876e5014cb8783bc63a24eeba
          b5e0ca75d547b51025ecfe75f4efadbc8d71c145c5
    + I * 1915fda1fb71039148f5d346f1c36df1630a2f908881f29de32a5c
          2782eb6eb3c8cbe58f8c1bf8d348319347c6ec7635
Q0.y    = 0e557684bd3e61db3f96df904c57ee1e8e45f5aecdda654ed74158
          7082ad91860d311cae158569c217c56bbba3d3f25f
    + I * 0acc1f70e15591005ab8bcdc7b1b19e3c16a6ee6c7a17ce83eaa27
          71971254be34726b266c076abab6b9b477ef790261
Q1.x    = 0cabb5826e6bf948e30cbb094b72685aa1d93ea49fdd9d54828b7f
          fb9df582e3d9405f33b9ae3ad3b6fd51863ff68c56
    + I * 0604d687830b1dba2cd28c644709475c7a5427aa15278df2db06f5
          9a48bcfb52061f77b6f5b637fc345e2bc64aa7e5ba
Q1.y    = 11e8a23425631218f3249f1870a8b1a17d82f3224602e433ff04a5
          525e827582ac5898e81972e21618e41e5c5edc03f5
    + I * 18f405a27aaf3803ef88b9f4d3e5d8eed901d980ebbefb71d5816a
          c2f1975c513965b7556ee44db2f74202ac178c72e4
```

G.10.2. BLS12381G2_XMD:SHA-256_SSWU_NU_

```
suite   = BLS12381G2_XMD:SHA-256_SSWU_NU_
dst     = BLS12381G2_XMD:SHA-256_SSWU_NU_TESTGEN

msg     =
P.x     = 170919c7845a9e623cef297e17484606a3eb2ae21ed8a21ff2b258
          861daefa3ac36955c0b374c6f4925868920d9c5f0b
    + I * 04264ddf941f7c9ea5ad62027c72b194c6c3f62a92fcdb56ddc9de
          7990489af1f81c576e7f451c2cd416102253e040f0
P.y     = 0ce03abe6c55ff0640b2b303440d88bd1a2b0cbfe3274b2802c1f5
          8b1085e4dd8795c9c4d9c166d2f033e3c438e7f8a9
    + I * 02d03d852629f70563e3a653ccc2e114439f551a2fd87c8136eb20
          5b84e22c3f40507beccdcdc52c921b69a57968ec7c
u[0]    = 09367e3b485dda3925e82cc458e5009051281d3e442e94f9ef9fee
          c44ee26375d6dc904dc1aa1f831f2aebd7b437ad12
    + I * 094376a68cdc8f64bd981d59bf762f9b2960df6b135f6e09ceada2
          fe8d0000bbf04023492796c09f8ef04016a2e8365f
Q.x     = 029b6b7335975135a3dd653cb5f865f8e1a6fd0e806f83f0807842
          6d294efb72578dc6747b81747d03b5bce9fa9c6d4c
    + I * 0e31914536a751dce017585d51c8c30127cf0abf8cce302faf8ea8
          7de1393a37696df8d999f597b256e8e19a0865817a
Q.y     = 0a718016d326692f10e7508c6abc624e1b37da5fd0e4391acf5a19
          fac36b97a9ae13a79dd2bd0b28c1db20f9d27607e9
    + I * 01df562b2fe9e7281b63a3c136a93773184ad924d9a0a0cd01b511
          50f175f9dfaa8d009f77df9812636a6de4a1b0a901

msg     = abc
P.x     = 16d830a4e12fddfbdaf9a667f94f21e490879fd3ccc5ee6f039cd7
          c2174fb47ea8027af78779a978d2a921612844587f
    + I * 019a3b47aa956b2b548cc04d9e109dec06642d6e28814f7e35f807
          e1ce609e2eae3a155af406c842529776d8192f562e
P.y     = 15930174c11aa9b51a5cc3ebfa1ab6377e2318c4ea2df387bdb84b
          28687a02c86e6401b195bbcabb6e95d6ae43669e12
    + I * 15adde069459ab2012b44c7703119185b96b7f04ad59b39f4f6aea
          35fdbb9c5c7d876b5f89afb55b67e7da96ad489dc3
u[0]    = 17ecd5d41a860b8886cb1210874b254f59945b089f774dcc14bc1a
          ca7d4e3c975bce0d28510c442e9a932be5880ee5b1
    + I * 0f105595e14847cc9a41fd70deb3240337678b266304100ec261ad
          d2585b991c7268bb1a325d2f871b327e8d04fd579b
Q.x     = 0f4c4441758b65035fab9adeb84dc4fcbf48e218085111cbc9e329
          4dde67ba93411d747d090fbc5aa144900df054ed32
    + I * 0bcf59db7917f351264cbc8825c04e88885b01f8228ecef238e39c
          d9d7e42c7a6b4aeeabacbfe43ea36f1a148c3517ae
Q.y     = 128fae582c1c32dc1199981981c9f2ff42343523192b82d3c010c6
          b06e419087449196cf4a79caa921db2ab10fc316b5
    + I * 0cff0768048fef20c10c19679deb85fff097befdb18a47a21c2ee5
          7eeebf046d4d9f3bdaff6f9c7d6c80e0700018b86e

msg     = abcdef0123456789
P.x     = 1498937f0ed18c49ebbcdee579b58ce235f3ab03be5dc809e1df25
```

```
              e2e0b4eb4c672f4eaf26df91f3755d6367df55d5be
        + I * 0910b2d55e210122fab2d2dae81e6a440fd22e925e422aaf16a8fd
              28477bacb12aa888de0faeea203e372a1c1cd9578c
P.y         = 033b1948575e70fed67fb4f7bd86b5452dfc0afeb74ecf5cab4a68
              72e33f0eade9564d3d5b9fcb9d4c498afda0bc037d
        + I * 102631eb4e684d759312d7eab78598f487c2c10ad3d3552cb43ce6
              f09a11eb46e551864863077906d3ecfd921f1fe541
u[0]        = 032ae17a23a76c94745a5460cd9f1191c0ebeec7adfc4df28b0833
              e536b7dbabf498dc076ff16cc11c6a6ef5105df693
        + I * 1107a6f450c6c9580c720190b577f52c633cf5f3defb528ae873d3
              723bccc8fa433014e9120a1da31abc27c674f37ae4
Q.x         = 110a8c50fb6b2df0146678e80de24089e0d619c45c488e0c688f13
              6963a4190b76647e9e122c18ab7b60a88ca1281e9c
        + I * 17f35d34544ad51d51f2ccfbda142addc678bf5551bf301dce3c3b
              934ecb6aa78b3814729282755a62e4680083736628
Q.y         = 0bd9c15e07f2a2bd5dacae74861afa19ea15d393fe7552d3ba45ef
              1396d37ff7967bdf67d93ea68baf849710bcd88147
        + I * 18e9a587c3b76f53a62e8919101b6f2b25803333a53b2c8596db44
              929bd59376f0a170c5debea9f8a378107c2ee1d51b

msg         = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x         = 18af6eedb7ed3be66c5a1d998ad4d9640f557b189558baec41f6e7
              12ff2a39f795a35494b4b12343b7a1a2b17686d793
        + I * 021f7faa0550e5a5d08338b4c0a5d30240dec7989fc7c77b6ffba9
              bfd5d64ce45af5aad8da8482bf0da91af4f29d371f
P.y         = 0cc46cea229960bfbe25831162c27f96cf8bb14c017938e35b6369
              87a306521915456fbd40633c6d5a30f61bce52a3f5
        + I * 166c1abec65af593d291dbd05e5d7d28f1a9ffb73751d65f49d760
              84493f3da707ee2bbf54cf6de5bbaac2ffa0028c31
u[0]        = 0cda6b874f8c41862c078099aa76d607be51d913a2e3f997539a09
              93bda31892292818c74aa9be035f234df2576fe49a
        + I * 0306162d24592a18fa8de2007d7b69d04bb7a71a5a7965d15bdcba
              a4ddf9b599079fbdae9f67d55ab6dba044f9daf179
Q.x         = 0cf7bef1339955cf2139e61d3876d22bfcc89c44492b458b2d08a6
              bb4fa58755739a4b216bfd7604e203f4c31cfcdc00
        + I * 0b80a0dee7817f29f8bf3374a3c033586d695391dc95280b2143d3
              98817bbf9b76547c54b81fd2b8ad24a6afa1dd8524
Q.y         = 07bba42c7be028acf9111e51cfa6d98f2949f12635847e938b0f79
              80705da9abd1d77bbdaf86f90512e0139ac0e57a6b
```

```
+ I * 0266340a1aaaf26dd5ec892f33dcb6a217dcde9b8d39dff5d277d9
      3130ee3e33e43a2cb83665fee8648d610da2967fa5
```

G.10.3. BLS12381G2_XMD:SHA-256_SVDW_RO_

```
suite   = BLS12381G2_XMD:SHA-256_SVDW_RO_
dst     = BLS12381G2_XMD:SHA-256_SVDW_RO_TESTGEN

msg     =
P.x     = 0d2b7cb4e1b0f001de23bda5465281418643463744 2e61bd7ae665
          f78e3e8429a3b0abf727b6ffc1a7d5d7f5683c2517
    + I * 0c6519468850b1b6b34c2ac7a67166d9c2b842df09105c8644d6d9
          be03880c56452e26dc82ee93d0db99946acc2675f4
P.y     = 063e54fd52fde42a5d4a0739ce89956deb0aa4721237b581a79af5
          e6e847dc047e0b0a41502975e4c15bed99cdcdf0e8
    + I * 0a04fe752f02d57217f2eca100582f9f60fe464aaec2e94624ff22
          a2522d562fc251e3e962b00f2a7cb544bad462487e
u[0]    = 178a86886823673336e71d5cb95ef38381506d64e1251fe3f66c2f
          ae08f5c1b5d1f01a0f09cbb29e8d776345c7601941
    + I * 1200915fc80bdf41b2723f5051a642fee4f548ebdd6f90da9d34d8
          f477fd17f84921be12497be94b9061bcc9e977a958
u[1]    = 0c2b3e012f715a94fc08bd3757ba7c979d848c9719264d2a6b07f0
          3be4f236da0f5017ee8b92fa4fac3ab88c64cce667
    + I * 099e1f36e1dada16e9277160ba74552efaa0c939a629c16fc8ebda
          75421af560a7ede7cd4423f00ffc1a12c07ab05fa0
Q0.x    = 0440b705d56079686eace2e3d2fc6c26ed4349ddb574bbb13ef5c9
          c25c7a757b43a1243ecfc62f9cdd169d6360c21cad
    + I * 061b98df942271495e7259a7010ed74c6bdf0f83fe225c7e60fa91
          b3699d32a99afa440f1ec380aacc8d703649631246
Q0.y    = 12964591a5a374d4ed6f1306e1889d8ab5259e88928cf0900c6f4f
          c64ca2cd7d1cc992166f0515c7ce53dd52efe3ded2
    + I * 1440790373a4369524884f589eff8c3ddeb82ad6032446072e75c3
          b09a300bfd9303a4794f6676ff2c73c796f9110557
Q1.x    = 01146e5fe16047ea58369ff4e6f53ada22c23260cb46b4ac1b24a7
          054691835c6f45e3a8218ed67c68a2c492b35a89b6
    + I * 196e511b97c1f3802e37478224bb5ce12082cee5dd8e5630ff194d
          8537c7b30ace746d39c051cf3f6a836a861f0fca71
Q1.y    = 16860436cd82bd22c166a92a38bae79036e357ce185ddcc481cef4
          6edf398d377b5d85f38325be63270cf09c31329583
    + I * 067e5d9c24201fa688bb9101b96fff337625778d0a6cb00495fa5a
          a8840bfed134b7b1bbd03979538b9b9bb5222b8a99

msg     = abc
P.x     = 0c2e41838e536c79576d6b34974204b591f0127354eac121b79029
          886a405615b273a6e1a78476d5d824d781b885af26
    + I * 0dca051f4d106f072564729f74969ab9e557760e14c67c55ed38fb
          7e2f3f4b26af1ce227c963fd06a5c5c2745f082415
P.y     = 12e8f2e906c3bac97820bd1cfad278d03321ba3d650a93cea80d4e
          ea70271aff8f145dc6d7c32e4e945b4a213a551871
    + I * 006d694e36dc47f3761ffc22d8a3cc66a9abe8851ad7dc42630a57
          3f569692d46de02256b9bf98f1066be5ce38d97836
u[0]    = 0ecb785eea491ba407b29812dd080692d7f654fa9be80e1b930f90
          a2764157ec6522aec0ebeca35a440b524b1efe2ade
    + I * 02f7de24231dbff773ec0ebdf9b1b7d84713ef0eff9ff8bc356163
```

```
          c34bc42e373abebb437568b9b8a7a9622f52f8f64d
u[1]    = 09ed2dafb8141db58cf0e3038c1b5969742a15dfd3a8de689309ad
          49477a8c4d45f0b3029d216bef8197a615b89ee53d
    + I * 0da9396afba6386ea945d8881d5d1b4e892ff506940d11f1c14e11
          008650abf458d6423185935f13e5304ac325996fed
Q0.x    = 056eff68c169148bf3cf4a1e3dcd47d05f172f54f0f0906759b98e
          e48897090c16127085c2f9a16dec9ef0b73e1e98ae
    + I * 0245cec38af51ed121d2d2f9ba21469335fb2eddc2c084386477c5
          3d2999c65182d43285464927b5f63465599955b283
Q0.y    = 1597b2110dc452b8083728a6ed061340b27cef4b2f96c7c4c74e3f
          090577407f2d8e72dd12e5a642eb0e0116d50d2900
    + I * 040a92bbe9662fa154a7706b0b208bbc5083e2909de5e0feb243c4
          aeb4c2abd9201d970fd586fe9fc3f7b70deeca38c0
Q1.x    = 0a231b6efadfd4cd9243b2d5f8d9820943f6616bd34dc48b677783
          f0d298211d06b9f25e7ebaf5c3d53009c99b2371bb
    + I * 0397e8e732c84a631df79723dc4ce49aeb1a08857cfd4272ca2b41
          d840862588fb75ac2555f127b3852942d5be5d8f32
Q1.y    = 0576ad544c7dff5cfb6fe52857e89f1ad5ee0f703826471a18c6e8
          a28f538a87de1d02cca1a4aa7c1a90f0f2e1efe9fa
    + I * 16eae9ad925c008d00832bad8830eaf05a2e693ddcf1fbb5d27025
          15fbbac163f0b12369e5466bcb30153616cf1a9021

msg     = abcdef0123456789
P.x     = 103b9ec29f230ce504d06b3d9efbebd5ef5cda2bbdbd09e66ed997
          9a84dd4ccbb86bdfdb4e9673b75af702cd933b9938
    + I * 17bb4ca24579cf87c16a554bc92497a67390971ad25d60a09befa7
          799b01676cdb30308ec0ddae9e4a5ca485200bc01f
P.y     = 161a632c2847e27715f550ea93ac9897dab5d7ad483cbf180efefa
          a232f2bba706bff16b1f1c3e0b9da9ff39e160e42f
    + I * 047f0b3db455080f13114bb2bbe37d40abdc9850bdc724f27a9ffa
          7bbd8307739fa1b803d69c30da64586c102842a3ae
u[0]    = 15788d0f014f083f4a6bad1e3ec01905fb81328fac060d575f9220
          ddb7fc495f6cca48e8f46c69153ef0152caa692bbe
    + I * 0c50eaf8096785caee1e08ff3d9d46b8e4c1a1600406d4ab8c9c96
          c74c4d2b6b90fb5ddb0bdf7adabe0b176f75005df7
u[1]    = 0274a965f4d5b3c5e1d35e426df580885eb9aeb4d0997afcb51d2a
          a908b7a7a5d2b608b4b054fbf77eb2ec8f6854d192
    + I * 07be630ece05ab3536328fc3ccbbc7b8af99542ce0a7241bcf0372
          3f9b45cbe16d3003dd28fcaaa4558f8f8f261a654a
Q0.x    = 0ad81631edcdf51ac0df8426a5d49192326f94244df93b8f990b9d
          23c2fd654b8f310a8bc0b136b649271a2ed598aeb7
    + I * 106a7d7e84ca907f28cdf2e2ab98a49d53aa70f512afc0c764fdff
          e778ff3538992546decf48142a790700c6eee5db70
Q0.y    = 0d9c4f07a23d596b3735a7e275a4a8ed3ac7b6849d83adf78b2743
          c6cb180232d5f180f7e32422c5e81039f9d48f9cb3
    + I * 0b4ee768a0d0525872db9e4c511da298691aa7a3360a38cfb89691
          da7c9a314a8d461e8cc8935d08b6fed60ef149e9d9
Q1.x    = 1351d14d9737ba5ded8fee48ee1e45c823eaec7b58600a5af7c354
          3c0b450c565db1e1d37732182dbcbbc31ea33e3e1f
```

```
    + I * 0c5ddcd88ebee0861a91dde563a6785752fbc5f4e079665ca2c056
          7dac8f203ed81f609e27e223938a55534ba8693657
Q1.y    = 146bbafa353ed101672c21e98fb2e1aa6f438c918d963b4f05e80c
          36cac3769ad25b49720b8566e1ea1eecff77b010cb
    + I * 02613eb3f365b783461d0edc9b3212495ae6f4de190150b94bbeca
          a6d631a6ccdf09335ac6f6aba57d70ce06d1a6d615

msg     = a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
          aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     = 125730d27604ffa5f1be4e6357f2dacf59803b6b8ce43b81eb2e42
          010e0765ea149ed52e5d8ad0847617bb87a3cfdbbc
    + I * 033008748ad6e6d95b68e86d8e786609c1011729606e45ab0b5691
          eceb3d4c72a80e36792f74e907309f3550bd7b9a6e
P.y     = 180de292e84a7ecc8361490a3f0d8bcf834e12d68529e437df7821
          11f01c8c73c53c9a502eac6aa9dffceae1b7df55b8
    + I * 07225c86be63f0b3eb8acf88ff4133fa9bb8c0b1d50c997ad33f57
          dcb9cd09cc4676da6f37fd7f4bfaa06bfca9bdb772
u[0]    = 00a73d9f99c991b8a95d533111700cb468bb9dfc7ecfa8879d278e
          fd4f62907dc735a137ed26eac6fbd4730d16c525e7
    + I * 011de2ac3ddd0fce82f81ff1506a4da615a545f5f1a38d76493489
          d043dec0298423741a607fd45b57526b2dbf7c0512
u[1]    = 054e5964345b7e40826826b6c0773ae205596eed3f430649873afb
          7ff36a0a4583d947b5e1dcaaf64b62067814724329
    + I * 0c022493d6de52b27b95ba1097b803e2834ce69555e25bda8df9a6
          0bbd7cb8ef2d11eea6ea85856626a1f35c2ccccb95
Q0.x    = 17dab789594c9400a35e44e124a04eb15a273b7ff3385aeb5767ce
          cf3a5d6ce03bee39e9b3d9eedef8dfc1c064465f04
    + I * 0d2b1ae115243ae13e9018e065bfb4215671e2ea86792e78858f01
          2264b642591f4839972f58c00309cbf54a6f2809c5
Q0.y    = 19d7a9d430474d353c6ccf18db52263cd2fa4685e0194b3ec55672
          dfea3645c08feb3643dc8a4e995406e3b1108e7275
    + I * 03ac4fde5674418babfbbc6bad412d789f018ad49135ef84e6cbb1
          b63eb9fb61d7d12caa9ccb4e710d1badff2fe47515
Q1.x    = 08d580532d837ce1cd78df4e3668b123eafd519b93e359ba64e028
          1740742649b76c960e5390e1fd4ea4abcc84afcaca
    + I * 12a30881e5bca0c8f3f8159b1490e215a380eb70c71f14570e886f
          6ba2de770e26ce6634e00c9a98e14665f61820209f
Q1.y    = 13192225f9eea2cb2c342e39654a115e5f4b002943cea6067d429e
          685317be5034edcfe71897728402fc28abb725eca9
    + I * 01c62e5d5325ec325e75e7e0a60a86d489ee543b0709ccd76807ca
          fd1d8041486c185a89f6ca72b96cb7eb193a3d5ef5
```

G.10.4. BLS12381G2_XMD:SHA-256_SVDW_NU_

```
suite   = BLS12381G2_XMD:SHA-256_SVDW_NU_
dst     = BLS12381G2_XMD:SHA-256_SVDW_NU_TESTGEN

msg     =
P.x     = 164c24901348f035811139a2ad95042bc85bb4b4481309431cd985
          03c951e9cb8f29d3c4ad0abeb31a3da4062f4b9027
    + I * 0cbca2904d96a263308df43e2767c4165f0357a5f0abf3419acab6
          fcf2200002b0b018c574f9253716844947f2752c94
P.y     = 12b642193b7beb7989fe98c06482effb8740d9e744ff317e050f75
          8bf449d6dedfded2eb1b1b3314b8699b9ea41f9fab
    + I * 0219ec9ba08594f6e1582fb6a6ffb795a92551b32191f146296d29
          497bbafba7b3aae8ab2f012a5780f72cb0d0380a78
u[0]    = 12e76b1034fcb8958d47b2ff763642841556e09d524a6e1ac14600
          9e7b0a60e859567d52629ea27abc86996632970e99
    + I * 005f69bf4eb6ef49bd04d4ca394c77b9ad359646e2ed36e013dc94
          91a64f2d1207734d4b91b53fd71a32d9e966d46dce
Q.x     = 033aa78402bfe8ee9117ac5501b55e2d1cc133a36ff2351b0176e7
          f44b009260f9f984be5aff18207a751fee8347250f
    + I * 0f6512f645a015b2917bcef9407299bc46e57344ec24877b681b2c
          7b3c8171bbd0efe0074fe5eaa7fc74983494f90521
Q.y     = 02ed21e9ce037653ba12b3996f085847db2daf80e0033013a67667
          c6c53983ee76af4946da08c7186164ed50d225551f
    + I * 05f95d63b6f2d45bc79824359754d23c795f98be958384829ea72b
          473525f58bef08bbf1c09ad153628f2d3ca9f44bdc

msg     = abc
P.x     = 14662fda486645b71a497576120b99cd0c8fd72c52ade4b13ce4de
          57ca05acba0facf6bdc1a230539a96e97fbfdef9fd
    + I * 0646f0fb608d82ba68075d6af75c61b762c5b47de2c620822e88ac
          d296fcfecf125113392b582c48c9c4cec645e7e817
P.y     = 097b57eed2bdcc07cf6518cc582af8604d284f51447d572f3c8b3c
          36c13f2f5ab3448d06a5e433a0a4753163469977b4
    + I * 0d0e9f7c5c51da1a8b3211052b0562b0f19690027d2bf0df3f662e
          c9743b0b39c6b9be5b5bf6d224f7eff946b3c4a149
u[0]    = 108fd37b239357d783a30c9aeea3129316f236cfa8e279bbd921ea
          7c642ffbb69e4f731cbee57b2df7b17e9ec19c8c91
    + I * 110587325ae905e360ad84368fe6aed5b8636bab77eaac3921c046
          8154c5d72ddd3ca2e9d57d393d55929344a34215c2
Q.x     = 065cca134296b189b4587c38490d674900cac3cb7a6e14ff96f415
          eae6ce4ee51696910182471d2542b86bc707a40230
    + I * 088edc31bb54b91389c1c057cc5b69e62eb06f72c3184834ce60bc
          df1ee885ad8e03005230bfb15c89109209d8c7671b
Q.y     = 17c0ef47c0c7960baec68b33d1ce4e9a54c2c770956cc43afe93d8
          b4bb07d5205fca8f785d55010111367b529a13a67b
    + I * 121a0b8aa8dc49fd8da99eb93d9d736e01e65a0e72c9144b3ba874
          16fdf125a355f38b5d98e98c20744cbae1c344386f

msg     = abcdef0123456789
P.x     = 0816a09925c45e39c3c04d3fdb331613f7308a0d5dec8e5496e4ae
```

```
              c5e5a67458532f25dd07c6a793bc4be8be17a9fd56
    + I *     0addce3d065de3f0e1b4ccad3e503b570412818094d1639329e7f4
              c6ad759bdcaff01f234ec3f1ebe71209a0773e4ca9
P.y     =     1214b5de271352309fd0091b72eecdb6224f58af9c76ae47a2b215
              12fe194d5c08278bbfbbd3468ba4ef3abbdb0ffcf0
    + I *     105bd74edfcf83e10663800fc4aca1f2de2e3197b6ad91c246c3c9
              828e24c44384747b95a8d156da7063df00546a6bad
u[0]    =     183b05a0844e652153f50ef0ea7e12bd9174707faf7fa3ac3d4408
              e8f06b0c8bae4aa322a71614f7d0380986632f2261
    + I *     011b40d28c312f5f62289fe083c493c84ac5c2e66f8be782c1d504
              93d30bab14fc88966dfe4d51f53049d1e7b221e0e7
Q.x     =     02da0a79d42b538d42ffc67176681d80ca8d0dc26974d3a30440ee
              92a362c13bbf737b414fbdf2fd8ce635a396e79ff0
    + I *     0e91dd04cd2d2e112e503efec26f84183fca55101c36fa41226c72
              eab8e398a189c9cf2f4eda77f7c9a38a09a563b2f1
Q.y     =     0d3d3ca0d4ec0aff793a69c22e2abf333e8efc5a72b1d5179c1947
              9e191daff895c95f32669672a1d3df38b1a184f8e8
    + I *     069ab87711da7286d7c75f8cdd4b1e2719554d7866139e948851bc
              14d7e9124f6f969118bdd2ae3db9d8f997b59717e2

msg     =     a512_aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
P.x     =     0eb76ebb9399768c509108314b557659a38bc8187e7ab742adf8c6
              72da98e2a7c9714e0e885070ccb7fbbabebf7a7bbb
    + I *     04049cf80dd5f3a2734402d6ef6ce67f35189a4ace622dae7cf02b
              8b9662cd58f05dde5b6920fb93c6126c5895e9a3a5
P.y     =     022d29c941c40481c8496f4b1df9fb708103c3170e99ffe41942e7
              9ce8b0e35ecdce0e9281da60a685bafe07f9a635a7
    + I *     023c57e63304ec8997a2b35ab78a2d060ffde49ff0235058ffdbf1
              29946672e518bb31506c53dad9c8a30b751b6181bf
u[0]    =     09619086497c2a6933c45e2d330c560562d87d3b2f77f1d2da5152
              5fb630a0ba2bd03ea9ebeb65778a65b29f1092a99d
    + I *     0c3c45a61a7938b2a8a645c48c37a2e2e957b5ef5bbe3f661e8bdc
              b50e962a548862335c8503d73c1d28c5dc598ea29f
Q.x     =     1764c36ebfdb830ad6bed870087c5969cc95e16ca57d572047f8bb
              17ba961d2264c86726ef69ce6122a4459a6025461c
    + I *     0d27880f35312ee235f2dc9208cc679679261e7f909072c5b71b14
              e369e20dc0c52459ace34c23aa6842e9b4663073e7
Q.y     =     07e98c5d7eadaa88299ee4ecccf848ca9492db725c45696596c500
              00cdf4852cee8d4a0e3ec0ca4d7b048ab0d92a4dee
```

```
+ I * 00b35c992495fa745f8a3b765a2e7d7ee06e7c4cdd6d26be78b062
      31a98cf9fd6b585d24b6f7a659c45e1acd64082af5
```

Authors' Addresses

Armando Faz-Hernandez
Cloudflare
101 Townsend St
San Francisco,
United States of America

Email: armfazh@cloudflare.com

Sam Scott
Cornell Tech
2 West Loop Rd
New York, New York 10044,
United States of America

Email: sam.scott@cornell.edu

Nick Sullivan
Cloudflare
101 Townsend St
San Francisco,
United States of America

Email: nick@cloudflare.com

Riad S. Wahby
Stanford University

Email: rsw@cs.stanford.edu

Christopher A. Wood
Apple Inc.
One Apple Park Way
Cupertino, California 95014,
United States of America

Email: cawood@apple.com