CFRG
Internet-Draft
Intended status: Informational
Expires: 5 August 2026

M. Abdalla
Nexus - San Francisco
B. Haase
Endress + Hauser Liquid Analysis - Gerlingen
J. Hesse
IBM Research Europe - Zurich

1 February 2026

CPace, a balanced composable PAKE
draft-irtf-cfrg-cpace-18

Abstract

This document describes CPace which is a protocol that allows two parties that share a low-entropy secret (password) to derive a strong shared key without disclosing the secret to offline dictionary attacks. The CPace protocol was tailored for constrained devices and can be used on groups of prime- and non-prime order.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Crypto Forum Research Group mailing list (cfrg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=cfrg.

Source for this draft and an issue tracker can be found at https://github.com/cfrg/draft-irtf-cfrg-cpace.

Status of This Memo

Table of Contents

1. Introduction

This document describes CPace which is a balanced Password-Authenticated-Key-Establishment (PAKE) protocol for two parties where both parties derive a cryptographic key of high entropy from a shared secret of low-entropy. CPace protects the passwords against offline dictionary attacks by requiring adversaries to actively interact with a protocol party and by allowing for at most one single password guess per active interaction.

The CPace design was tailored considering the following main objectives:

* Efficiency: Deployment of CPace is feasible on resource-constrained devices.

* Versatility: CPace supports different application scenarios via versatile input formats, and by supporting applications with and without clear initiator and responder roles.

* Implementation error resistance: CPace aims at avoiding common implementation pitfalls already by design, such as avoiding incentives for insecure execution-time speed optimizations. For smooth integration into different cryptographic library ecosystems, this document provides a variety of cipher suites.

* Post-quantum annoyance: CPace comes with measures to slow down adversaries capable of breaking the discrete logarithm problem on elliptic curves.

1.1. Outline of this document

* Section 3 describes the expected properties of an application using CPace, and discusses in particular which application-level aspects are relevant for CPace's security.

* Section 4 gives an overview of the recommended cipher suites for CPace which were optimized for different types of cryptographic library ecosystems.

* Section 5 introduces the notation used throughout this document.

* Section 6 specifies the CPace protocol.

* The appendix provides code and test vectors of all of the functions defined for CPace.

As this document is primarily written for implementers and application designers, we would like to refer the theory-inclined reader to the scientific paper [AHH21] which covers the detailed security analysis of the different CPace instantiations as defined in this document via the cipher suites.

2. Requirements Notation

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

3. High-level application perspective

CPace enables balanced password-authenticated key establishment. I.e. with CPace the parties start the protocol with a shared secret octet string, namely the password-related string (PRS). PRS can be a low-entropy secret itself, for instance a clear-text password encoded according to [RFC8265], or any string derived from a common secret, for instance by use of a key derivation function.

Applications with clients and servers where the server side is storing account and password information in its persistent memory are recommended to use _augmented_ PAKE protocols such as OPAQUE [I-D.irtf-cfrg-opaque].

In the course of the CPace protocol, A sends one message to B and B sends one message to A. We use the term "initiator-responder" for CPace where A always speaks first, and the term "symmetric" setting where anyone can speak first.

CPace’s output is an intermediate session key (ISK), but any party might abort in case of an invalid received message. A and B will produce the same ISK value if and only if both sides did initiate the protocol using the same protocol inputs, specifically the same PRS and the same value for the input parameters CI, ADa, ADb and sid that will be specified in section Section 3.1.

This specification considers different application scenarios. This includes applications aiming at anonymous key exchange and applications that need to rely on verification of identities of one or both communication partners. Moreover, when identities are used, they may or may not need to be kept confidential. Depending on the

application’s requirements, identity information regarding the communication partners may have to be mandatorily integrated in the input parameters CI, ADa, ADb and the protocol may have to be executed with clear initiator and responder roles (see Section 9.1).

The naming of ISK as "intermediate" session key highlights the fact that it is RECOMMENDED that applications process ISK by use of a suitable strong key derivation function KDF (such as defined in [RFC5869]) before using the key in a higher-level protocol.

3.1. CPace inputs

For accommodating different application settings, CPace offers the following inputs which, depending on the application scenario, MAY also be the empty string:

* Party identity strings (A,B). In CPace, each party can be given a party identity string which might be a device name, a user name, or an URL. CPace offers two alternative options for authenticating the party identifiers in the course of the protocol run (see Section 9.1).

* Channel identifier (CI). CI can be used to bind a session key exchanged with CPace to a specific networking channel which interconnects the protocol parties. CI could for instance include networking addresses of both parties or party identity strings or service port number identifiers. Both parties are required to have the same view of CI. CI will not be publicly sent on the wire and may also include confidential information. Both parties will only establish a common session key if they initiated the protocol with the same view of CI.

* Associated data fields (ADa and ADb). These fields can be used for authenticating associated data alongside the CPace protocol. The ADa and ADb will be sent in clear text as part of the protocol messages. ADa and ADb will become authenticated in a CPace protocol run as both parties will only agree on a common key if they have had the same view on ADa and ADb. Applications that need to rely on the identity of the communication partner may have to integrate identity information in ADa and/or ADb (see Section 9.1).

  In a setting with clear initiator and responder roles, identity information in ADa sent by the initiator can be used by the responder for choosing the appropriate PRS (respectively password) for this identity. ADa and ADb could also include application protocol version information (e.g. to avoid downgrade attacks).

* Session identifier (sid). If both parties have access to the same unique public octet string sid being specific for a communication session before starting the protocol, it is RECOMMENDED to use this sid value as an additional input for the protocol as this provides security advantages and will bind the CPace run to this communication session (see Section 9).

3.2. Optional CPace outputs

If a session identifier is not available as input at protocol start CPace can optionally produce a unique public session identifier sid_output as output that might be helpful for the application for actions subsequent to the CPace protocol step (see Section 9.6, [BGHJ24]).

3.3. Responsibilities of the application layer

The following tasks are out of the scope of this document and left to the application layer

* Setup phase: The application layer is responsible for the handshake that makes parties agree on a common CPace cipher suite.

* This document does not specify which encodings applications use for the mandatory PRS input and the inputs CI, sid, ADa and ADb. If PRS is a clear-text password or an octet string derived from a clear-text password, e.g. by use of a key-derivation function, the clear-text password SHOULD BE encoded according to [RFC8265].

* The application needs to settle whether CPace is used in the initiator-responder or the symmetric setting along the guidelines of Section 9.1. In the symmetric setting, transcripts ordered string concatenation must be used for generating protocol transcripts. In this document we will provide test vectors for both the initiator-responder and the symmetric setting.

4. CPace cipher suites

In the setup phase of CPace, both communication partners need to agree on a common cipher suite. Cipher suites consist of a combination of a hash function H and an elliptic curve environment G.

For naming cipher suites we use the convention "CPACE-G-H". We RECOMMEND the following cipher suites:

* CPACE-X25519-SHA512. This suite uses the group environment G_X25519 defined in Section 7.2 and SHA-512 as hash function. This cipher suite comes with the smallest messages on the wire and a low computational cost.

* CPACE-P256_XMD:SHA-256_SSWU_NU_-SHA256. This suite instantiates the group environment G as specified in Section 7.4 using the encode_to_curve function P256_XMD:SHA-256_SSWU_NU_ from [RFC9380] on curve NIST-P256, and hash function SHA-256.

The following RECOMMENDED cipher suites provide higher security margins.

* CPACE-X448-SHAKE256. This suite uses the group environment G_X448 defined in Section 7.2 and SHAKE-256 as hash function.

* CPACE-P384_XMD:SHA-384_SSWU_NU_-SHA384. This suite instantiates G as specified in Section 7.4 using the encode_to_curve function P384_XMD:SHA-384_SSWU_NU_ from [RFC9380] on curve NIST-P384 with H = SHA-384.

* CPACE-P521_XMD:SHA-512_SSWU_NU_-SHA512. This suite instantiates G as specified in Section 7.4 using the encode_to_curve function P521_XMD:SHA-512_SSWU_NU_ from [RFC9380] on curve NIST-P521 with H = SHA-512.

CPace can also securely be implemented using the cipher suites CPACE-RISTR255-SHA512 and CPACE-DECAF448-SHAKE256 defined in Section 7.3. Section 9 gives guidance on how to implement CPace on further elliptic curves.

5. Definitions and notation

5.1. Group environment G

The group environment G specifies an elliptic curve group (also denoted G for convenience) and associated constants and functions as detailed below. In this document we use additive notation for the group operation.

* G.calculate_generator(H,PRS,CI,sid) denotes a function that outputs a representation of a generator (referred to as "generator" from now on) of the group which is derived from input octet strings PRS, CI, and sid and with the help of a hash function H.

* G.sample_scalar() is a function returning a representation of an integer (referred to as "scalar" from now on) appropriate as a private Diffie-Hellman key for the group.

* G.scalar_mult(y,g) is a function operating on a scalar y and a group element g. It returns an octet string representation of the group element Y = y*g.

* G.I denotes a unique octet string representation of the neutral element of the group. G.I is used for detecting and signaling certain error conditions.

* G.scalar_mult_vfy(y,g) is a function operating on a scalar y and a group element g. It returns an octet string representation of the group element y*g. Additionally, scalar_mult_vfy specifies validity conditions for y,g and y*g and outputs G.I in case they are not met.

* G.DSI denotes a domain-separation identifier octet string which SHALL be uniquely identifying the group environment G.

5.2. Hash function H

Common choices for H are SHA-512 [RFC6234] or SHAKE-256 [FIPS202]. (I.e., the hash function outputs octet strings, and not group elements.) For considering both variable-output-length hashes and fixed-output-length hashes, we use the following convention.

We use the following notation for referring to the specific properties of a hash function H:

* H.hash(m,l) is a function that operates on an input octet string m and returns the first l octets of the hash of m.

* H.b_in_bytes denotes the _minimum_ output size in bytes for collision resistance for the security level target of the hash function. E.g. H.b_in_bytes = 64 for SHA-512 and SHAKE-256 and H.b_in_bytes = 32 for SHA-256 and SHAKE-128. We use the notation H.hash(m) = H.hash(m, H.b_in_bytes) and let the hash operation output the default length if no explicit length parameter is given.

* H.bmax_in_bytes denotes the _maximum_ output size in octets supported by the hash function. In case of fixed-size hashes such as SHA-256, this is the same as H.b_in_bytes, while there is no such limit for hash functions such as SHAKE-256.

* H.s_in_bytes denotes the _input block size_ used by H. This number denotes the maximum number of bytes that can be processed in a single block before applying the compression function or permutation becomes necessary. (See also [RFC2104] for the corresponding block size concepts). For instance, for SHA-512 the input block size s_in_bytes is 128 as the compression function can process up to 128 bytes, while for SHAKE-256 the input block size amounts to 136 bytes before the permutation of the sponge state needs to be applied.

5.3. Notation for string operations

* bytes1 || bytes2 denotes concatenation of octet strings.

* len(S) denotes the number of octets in an octet string S.

* This document uses quotation marks "" both for general language (e.g. for citation of notation used in other documents) and as syntax for specifying octet strings as in b"CPace25519".

  We use a preceding lowercase letter b"" in front of the quotation marks if a character sequence is representing an octet string sequence. I.e., we use the notation convention for byte string representations with single-byte ASCII character encodings from the python programming language. b"" denotes the empty string of length 0.

* LEB128 denotes an algorithm that converts an integer to a variable sized string. The algorithm encodes 7 bits per byte starting with the least significant bits in bits #0 to #6. As long as significant bits remain, bit #7 will be set. This will result in a single-byte encoding for values below 128. Test vectors and reference code for LEB128 encoding are available in the appendix.

* prepend_len(octet_string) denotes the octet sequence that is obtained from prepending the length of the octet string to the string itself. The length is encoded using LEB128. Test vectors and code for prepend_len are available in the appendix.

* lv_cat(a0,a1, ...) is the "length-value" encoding function which returns the concatenation of the input strings with an encoding of their respective length prepended. E.g., lv_cat(a0,a1) returns prepend_len(a0) || prepend_len(a1). The detailed specification of lv_cat and reference code is available in the appendix.

* sample_random_bytes(n) denotes a function that returns n octets, each of which is to be independently sampled from a uniform distribution between 0 and 255.

* zero_bytes(n) denotes a function that returns n octets with value 0.

* o_cat(bytes1,bytes2) denotes a function for ordered concatenation of octet strings. It places the lexicographically larger octet string first and prepends the two bytes from the octet string b"oc" to the result. Reference code for this function is available in the appendix.

* transcript(Ya,ADa, Yb,ADb) denotes a function outputting an octet string for the protocol transcript. In applications where CPace is used without clear initiator and responder roles, i.e. where the ordering of messages is not enforced by the protocol flow, transcript_oc(Ya,ADa, Yb,ADb) = o_cat(lv_cat(Ya,ADa),lv_cat(Yb, ADb)) SHALL be used. In the initiator-responder setting, the implementation transcript_ir(Ya,ADa, Yb,ADb) = lv_cat(Ya,ADa) || lv_cat(Yb, ADb) SHALL be used.

5.4. Notation for group operations

We use additive notation for the group, i.e., 2*X denotes the element that is obtained by computing X+X, for group element X and group operation +.

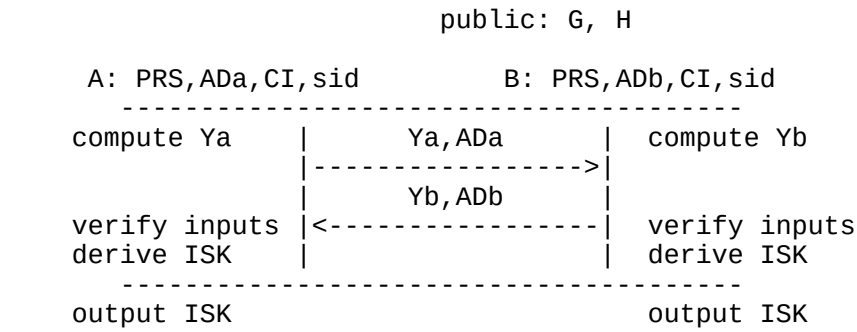
6. The CPace protocol

CPace is a one round protocol between two parties, A and B. At invocation, A and B are provisioned with PRS,G and H. Parties will also be provisioned with CI,sid,ADa (for A) and CI,sid,ADb (for B) which will default to the empty string b"" if not used.

Party identifiers SHALL be integrated into CI and/or ADa and ADb following the guidelines in Section 9.1.

A sends the public share Ya and associated data ADa to B. Likewise, B sends the public share Yb and associated data ADb to A. Both A and B use the received messages for deriving a shared intermediate session key, ISK.

6.1. Protocol flow



6.2. CPace protocol instructions

A computes a generator g = G.calculate_generator(H,PRS,CI,sid), scalar ya = G.sample_scalar() and group element Ya = G.scalar_mult (ya,g). A then transmits Ya and associated data ADa to B.

B computes a generator g = G.calculate_generator(H,PRS,CI,sid), scalar yb = G.sample_scalar() and group element Yb = G.scalar_mult(yb,g). B sends Yb and associated data ADb to A.

B then computes K = G.scalar_mult_vfy(yb,Ya). B MUST abort if K=G.I. Otherwise B calculates ISK = H.hash(lv_cat(G.DSI || b"_ISK", sid, K)||transcript(Ya,ADa,Yb,ADb)). B returns ISK and terminates.

Likewise upon reception of the message from B, A computes K = G.scalar_mult_vfy(ya,Yb). A MUST abort if K=G.I.

Otherwise A calculates ISK = H.hash(lv_cat(G.DSI || b"_ISK", sid, K) || transcript(Ya,ADa,Yb,ADb)). A returns ISK and terminates.

The session key ISK returned by A and B is identical if and only if the supplied input parameters PRS, CI and sid match on both sides and the transcript views of both parties match.

7. Implementation of recommended CPace cipher suites

7.1. Common function for computing generators

The different cipher suites for CPace defined in the upcoming sections share the following method for deterministically combining the individual strings PRS, CI, sid and the domain-separation identifier DSI to a generator string:

* generator_string(DSI, PRS, CI, sid, s_in_bytes) denotes a function that returns the string lv_cat(DSI, PRS, zero_bytes(len_zpad), CI, sid).

* len_zpad = MAX(0, s_in_bytes - len(prepend_len(PRS)) - len(prepend_len(G.DSI)) - 1)

The zero padding of length len_zpad is designed such that the encoding of DSI and PRS together with the zero padding field completely fills at least the first input block (of length s_in_bytes) of the hash. As a result for the common case of short PRS the number of bytes to hash becomes independent of the actual length of the password (PRS). (Code and test vectors are provided in the appendix.)

The introduction of a zero-padding within the generator string also helps mitigating attacks of a side-channel adversary that analyzes correlations between publicly known variable information with a short low-entropy PRS string. Note that the hash of the first block is intentionally made independent of session-specific inputs, such as sid or CI and that there is no limitation regarding the maximum length of the PRS string.

7.2. CPace group objects G_X25519 and G_X448 for single-coordinate Ladders on Montgomery curves

In this section we consider the case of CPace when using the X25519 and X448 Diffie-Hellman functions from [RFC7748] operating on the Montgomery curves Curve25519 and Curve448 [RFC7748]. CPace implementations using single-coordinate ladders on further Montgomery curves SHALL use the definitions in line with the specifications for X25519 and X448 and review the guidance given in Section 9.

For the group environment G_X25519 the following definitions apply:

* G_X25519.field_size_bytes = 32

* G_X25519.field_size_bits = 255

* G_X25519.sample_scalar() = sample_random_bytes(G.field_size_bytes)

* G_X25519.scalar_mult(y,g) = G.scalar_mult_vfy(y,g) = X25519(y,g)

* G_X25519.I = zero_bytes(G.field_size_bytes)

* G_X25519.DSI = b"CPace255"

CPace cipher suites using G_X25519 MUST use a hash function producing at least H.b_max_in_bytes >= 32 bytes of output. It is RECOMMENDED to use G_X25519 in combination with SHA-512.

For X448 the following definitions apply:

* G_X448.field_size_bytes = 56

* G_X448.field_size_bits = 448

* G_X448.sample_scalar() = sample_random_bytes(G.field_size_bytes)

* G_X448.scalar_mult(y,g) = G.scalar_mult_vfy(y,g) = X448(y,g)

* G_X448.I = zero_bytes(G.field_size_bytes)

* G_X448.DSI = b"CPace448"

CPace cipher suites using G_X448 MUST use a hash function producing at least H.b_max_in_bytes >= 56 bytes of output. It is RECOMMENDED to use G_X448 in combination with SHAKE-256.

For both G_X448 and G_X25519 the G.calculate_generator(H, PRS,sid,CI) function shall be implemented as follows.

* First gen_str = generator_string(G.DSI,PRS,CI,sid, H.s_in_bytes) SHALL BE calculated using the input block size of the chosen hash function.

* This string SHALL then BE hashed to the required length gen_str_hash = H.hash(gen_str, G.field_size_bytes). Note that this implies that the permissible output length H.maxb_in_bytes MUST BE larger or equal to the field size of the group G for making a hashing function suitable.

* This result is then considered as a field coordinate using the u = decodeUCoordinate(gen_str_hash, G.field_size_bits) function from [RFC7748] which we repeat in the appendix for convenience.

* The result point g is then calculated as (g,v) = map_to_curve_elligator2(u) using the function from [RFC9380]. Note that the v coordinate produced by the map_to_curve_elligator2 function is not required for CPace and discarded. The appendix repeats the definitions from [RFC9380] for convenience.

Code for the functions above is available in the appendix.

7.2.1. Verification tests

For single-coordinate Montgomery ladders on Montgomery curves, verification tests according to Section 8 SHALL check for proper handling of the abort conditions, when a party is receiving u coordinate values that encode a low-order point on either the curve or the quadratic twist.

In addition to that, in case of G_X25519, the tests SHALL also verify that the implementation of G.scalar_mult_vfy(y,g) produces the expected results for non-canonical u coordinate values with bit #255 set, which may also encode low-order points.

Corresponding test vectors are provided in the appendix.

7.3. CPace group objects G_Ristretto255 and G_Decaf448 for prime-order group abstractions

In this section we consider the case of CPace using the Ristretto255 and Decaf448 group abstractions [I-D.draft-irtf-cfrg-ristretto255-decaf448]. These abstractions define an encode and decode function, group operations using an internal encoding and an element-derivation function that maps a byte string to a group element. With the group abstractions there is a distinction between an internal representation of group elements and an external encoding of the same group element. In order to distinguish between these different representations, we prepend an underscore before values using the internal representation within this section.

For Ristretto255 the following definitions apply:

* G_Ristretto255.DSI = b"CPaceRistretto255"

* G_Ristretto255.field_size_bytes = 32

* G_Ristretto255.group_size_bits = 252

* G_Ristretto255.group_order = 2^252 + 27742317777372353535851937790883648493

CPace cipher suites using G_Ristretto255 MUST use a hash function producing at least H.b_max_in_bytes >= 64 bytes of output. It is RECOMMENDED to use G_Ristretto255 in combination with SHA-512.

For decaf448 the following definitions apply:

* G_Decaf448.DSI = b"CPaceDecaf448"

* G_Decaf448.field_size_bytes = 56

* G_Decaf448.group_size_bits = 445

* G_Decaf448.group_order = l = 2^446 - 138180668098951153520073867485154268803366924748821786098945475038 85

CPace cipher suites using G_Decaf448 MUST use a hash function producing at least H.b_max_in_bytes >= 112 bytes of output. It is RECOMMENDED to use G_Decaf448 in combination with SHAKE-256.

For both abstractions the following definitions apply:

* It is RECOMMENDED to implement G.sample_scalar() as follows.
  - Set scalar = sample_random_bytes(G.group_size_bytes).
  - Then clear the most significant bits larger than G.group_size_bits.
  - Interpret the result as the little-endian encoding of an integer value and return the result.
* Alternatively, if G.sample_scalar() is not implemented according to the above recommendation, it SHALL be implemented using uniform sampling between 1 and (G.group_order - 1). Note that the more complex uniform sampling process can provide a larger side-channel attack surface for embedded systems in hostile environments.
* G.scalar_mult(y,_g) SHALL operate on a scalar y and a group element _g in the internal representation of the group abstraction environment. It returns the value Y = encode(y * (_g)), i.e. it returns a value using the public encoding.
* G.I = is the public encoding representation of the identity element.
* G.scalar_mult_vfy(y,X) operates on a value using the public encoding and a scalar and is implemented as follows. If the decode(X) function fails, it returns G.I. Otherwise it returns encode(y * decode(X)).
* The G.calculate_generator(H, PRS,sid,CI) function SHALL return a decoded point and SHALL BE implemented as follows.
  - First gen_str = generator_string(G.DSI,PRS,CI,sid, H.s_in_bytes) is calculated using the input block size of the chosen hash function.
  - This string is then hashed to the required length gen_str_hash = H.hash(gen_str, 2 * G.field_size_bytes). Note that this implies that the permissible output length H.maxb_in_bytes MUST BE larger or equal to twice the field size of the group G for making a hash function suitable.

- Finally the internal representation of the generator _g is calculated as _g = element_derivation(gen_str_hash) using the element derivation function from the abstraction.

Note that with these definitions the scalar_mult function operates on a decoded point _g and returns an encoded point, while the scalar_mult_vfy(y,X) function operates on an encoded point X (and also returns an encoded point).

7.3.1. Verification tests

For group abstractions verification tests according to Section 8 SHALL check for proper handling of the abort conditions, when a party is receiving encodings of the neutral element or receives an octet string that does not decode to a valid group element.

7.4. CPace group objects for curves in Short-Weierstrass representation

The group environment objects G defined in this section for use with Short-Weierstrass curves, are parametrized by the choice of an elliptic curve and by choice of a suitable encode_to_curve function. encode_to_curve must map an octet string to a point on the curve.

7.4.1. Curves and associated functions

Elliptic curves in Short-Weierstrass form are considered in [IEEE1363]. [IEEE1363] allows for both, curves of prime and non-prime order. However, for the procedures described in this section any suitable group MUST BE of prime order.

The specification for the group environment objects specified in this section closely follow the ECKAS-DH1 method from [IEEE1363]. I.e. we use the same methods and encodings and protocol sub steps as employed in the TLS [RFC5246] [RFC8446] protocol family.

For CPace only the uncompressed full-coordinate encodings from [SEC1] (x and y coordinate) SHOULD be used. Commonly used curve groups are specified in [SEC2] and [RFC5639]. A typical representative of such a Short-Weierstrass curve is NIST-P256. Point verification as used in ECKAS-DH1 is described in Annex A.16.10. of [IEEE1363].

For deriving Diffie-Hellman shared secrets ECKAS-DH1 from [IEEE1363] specifies the use of an ECSVDP-DH method. We use ECSVDP-DH in combination with the identity map such that it either returns "error" or the x-coordinate of the Diffie-Hellman result point as shared secret in big endian format (fixed length output by FE2OSP without truncating leading zeros).

7.4.2. Suitable encode_to_curve methods

All the encode_to_curve methods specified in [RFC9380] are suitable for CPace. For Short-Weierstrass curves it is RECOMMENDED to use the non-uniform variant of the SSWU mapping primitive from [RFC9380] if a SSWU mapping is available for the chosen curve. (We recommend non-uniform maps in order to give implementations the flexibility to opt for x-coordinate-only scalar multiplication algorithms.)

7.4.3. Definition of the group environment G for Short-Weierstrass curves

In this paragraph we use the following notation for defining the group object G for a selected curve and encode_to_curve method:

* With G.group_order we denote the order of the elliptic curve which MUST BE a prime.

* With is_valid(X) we denote a method which operates on an octet stream according to [SEC1] of a point on the group and returns true if the point is valid and returns false otherwise. This is_valid(X) method SHALL be implemented according to Annex A.16.10. of [IEEE1363]. I.e. it shall return false if X encodes either the neutral element on the group or does not form a valid encoding of a point on the group.

* With encode_to_curve(str,DST) we denote a mapping function from [RFC9380]. I.e. a function that maps octet string str to a point on the group using the domain separation tag DST. [RFC9380] considers both, uniform and non-uniform mappings based on several different strategies. It is RECOMMENDED to use the nonuniform variant of the SSWU mapping primitive within [RFC9380].

* G.DSI denotes a domain-separation identifier octet string. G.DSI which SHALL BE obtained by the concatenation of b"CPace" and the associated name of the cipher suite used for the encode_to_curve function as specified in [RFC9380]. E.g. when using the map with the name P384_XMD:SHA-384_SSWU_NU_ on curve NIST-P384 the resulting value SHALL BE G.DSI = b"CPaceP384_XMD:SHA-384_SSWU_NU_".

Using the above definitions, the CPace functions required for the group object G are defined as follows.

* G.DST denotes the domain-separation tag value to use in conjunction with the encode_to_curve function from [RFC9380]. G.DST shall be obtained by concatenating G.DSI and b"_DST".

* G.sample_scalar() SHALL return a value between 1 and (G.group_order - 1). The sampling SHALL BE indistinguishable from uniform random selection between 1 and (G.group_order - 1). It is RECOMMENDED to use a constant-time rejection sampling algorithm for converting a uniform bitstring to a uniform value between 1 and (G.group_order - 1).

* G.calculate_generator(H, PRS,sid,CI) function SHALL be implemented as follows.

  - First gen_str = generator_string(G.DSI,PRS,CI,sid, H.s_in_bytes) is calculated.

  - Then the output of a call to encode_to_curve(gen_str, G.DST) is returned, using the selected suite from [RFC9380].

* G.scalar_mult(s,X) is a function that operates on a scalar s and an input point X. The input X shall use the same encoding as produced by the G.calculate_generator method above. G.scalar_mult(s,X) SHALL return an encoding of either the point s*X or the point (-s)*X according to [SEC1]. Implementations SHOULD use the full-coordinate format without compression, as important protocols such as TLS 1.3 removed support for compression. Implementations of scalar_mult(s,X) MAY output either s*X or (-s)*X as both points s*X and (-s)*X have the same x-coordinate and result in the same Diffie-Hellman shared secrets K. (This allows implementations to opt for x-coordinate-only scalar multiplication algorithms.)

* G.scalar_mult_vfy(s,X) merges verification of point X according to [IEEE1363] A.16.10. and the the ECSVDP-DH procedure from [IEEE1363]. It SHALL BE implemented as follows:

  - If is_valid(X) = False then G.scalar_mult_vfy(s,X) SHALL return "error" as specified in [IEEE1363] A.16.10 and 7.2.1.

  - Otherwise G.scalar_mult_vfy(s,X) SHALL return the result of the ECSVDP-DH procedure from [IEEE1363] (section 7.2.1). I.e. it shall either return "error" (in case that s*X is the neutral element) or the secret shared value "z" defined in [IEEE1363] (otherwise). "z" SHALL be encoded by using the big-endian encoding of the x-coordinate of the result point s*X according to [SEC1].

* We represent the neutral element G.I by using the representation of the "error" result case from [IEEE1363] as used in the G.scalar_mult_vfy method above.

7.4.4. Verification tests

For Short-Weierstrass curves verification tests according to Section 8 SHALL check for proper handling of the abort conditions, when a party is receiving an encoding of the point at infinity and an encoding of a point not on the group.

8. Implementation verification

Any CPace implementation MUST be tested against invalid or weak point attacks. Implementation MUST be verified to abort upon conditions where G.scalar_mult_vfy functions outputs G.I. For testing an implementation it is RECOMMENDED to include weak or invalid point encodings within the messages of party A and B and introduce this in a protocol run. It SHALL be verified that the abort condition is properly handled.

Corresponding test vectors are given in the appendix for all recommended cipher suites.

9. Security Considerations

A security proof of CPace is found in [AHH21]. This proof covers all recommended cipher suites included in this document. The security analysis in [BGHJ24] extends the analysis from [AHH21] by covering the case that no pre-agreed session identifier is available. [BGHJ24] also shows how a unique session id sid_output can be generated along with the protocol for applications that do not have a session identifier input available.

9.1. Party identifiers

The protocol assures that both communication partners have had the same view on the communication transcripts and the inputs CI and sid.

If CPace is instantiated without identity strings for A or B in its inputs it will anonymously create a key with any party using the same PRS and sid values and cannot give any further guarantee regarding the identity of the communication partner. A protocol instance running on a party P might even be communicating with a second protocol instance also running on P.

9.1.1. Guidance regarding party identifier integration

If an application layer’s security relies on CPace for checking party identities, it SHALL integrate the party identifiers that are to be checked in the CPace protocol run within CI or ADa/ADb as specified below.

* If CPace is used in initiator-responder mode, identity strings that are to be authenticated and that are available for both communication partners at protocol start SHOULD be integrated as part of CI.

  If both party identifiers are integrated into CI, the encoding MUST associate the initiator and responder roles with the respective identity strings. It is recommended to place the initiator‘s identity first and the responder‘s identity second.

  Party identity information included in CI will be kept confidential.

* Party identities that are not included in CI identity and are to be authenticated by CPace SHALL be integrated in ADa/ADb, such that A integrates its party identifier in ADa and B integrates its party identifier in ADb. In this case, the application layer SHALL make the recipient check the party identifier string of the remote communication partner.

  Note that identities communicated in ADa or ADb will not be kept confidential.

If ADa and ADb are not guaranteed to be unique, then CPace SHALL be used in initator-responder mode.

If CPace is to be run in the symmetric mode without initiator and responder roles, the application can always enforce uniqueness of ADa and ADb for all sessions by adding further information such as random data.

9.1.2. Rationale for the above guidance

Incorporating party identifier strings is important for fending off attacks based on relaying messages. Such attacks become for example relevant in a setting where several parties, say, A, B and C, share the same password PRS. An adversary might relay messages from an honest user A, who aims at interacting with user B, to a party C instead. If no party identifier strings are used and B and C share the same PRS value, then A might be using CPace for establishing a common ISK key with C while assuming to interact with party B. If a party A is allowing for multiple concurrent sessions, the adversary may also mount an attack relaying messages of A looped back to A such that A actually shares a key with itself [HS14].

Integration of party identity strings in CI is to be preferred. This way, the identities may be kept confidential. If both identities are to be integrated in CI, this is only possible if clear initiator and responder roles are assigned and the encoding of the identities associates the role with the identity string.

Integration of identity strings in CI also avoids the need of the security-critical subsequent check for the identity strings, which might be omitted or implemented incorrectly without notice. Integration of identities into CI also strengthens the security properties with respect to attacks based on quantum computers Section 9.12.

Applications that integrate identity strings in ADa and/or ADb shall carefully verify implementations for correctness of the implemented identity checks that the application must carry out after the CPace run.

When adding randomness guaranteeing for unique values of ADa and ADb then a party running the application can detect for loopback attacks by checking that the received remote value of ADa/ADb doesn’t show up in the list of active local concurrent protocol sessions [HMSD18].

If no unique value in ADa and ADb is available or if maintaining state information regarding the list of concurrently active local protocol instances for verification is impractical in a given application setting, then the loopback attack may be prevented by assigning initiator and responder role and mandating that a given party implements either the initiator or responder role for a given PRS password but not both roles with the same (PRS,sid) value set.

Note that the requirement on party identifiers may differ from what might be intuitively expected as information on the application service such as service identifiers, port-ids and role information (e.g. client or server role) should be included as part of the party identity.

For instance, if computers A and B allow for running a protocol with different roles (e.g. both might run several client and a server instances concurrently on different ports) then a relay attack may successfully generate protocol confusion. E.g. a client instance on A may be maliciously redirected to a second client instance on B while it expects to be connecting to a server on B. This will work if client and server instances on B share the same PRS secret and the identity strings do not include information on the respective roles.

9.2. Hashing protocol transcripts

CPace prepends the length of all variable-size input strings before hashing data. Prepending the length of all variable-size input strings results in a so-called prefix-free encoding of transcript strings, using terminology introduced in [CDMP05]. This property allows for disregarding length-extension imperfections that come with the commonly used Merkle-Damgard hash function constructions such as SHA256 and SHA512 [CDMP05].

9.3. Key derivation

A CPace implementation SHALL output ISK but MUST NOT expose K, because a leaked K may enable offline dictionary attack on the password, and a matching value for K does not provide authentication of ADa and ADb.

As noted already in Section 6 it is RECOMMENDED to process ISK by use of a suitable strong key derivation function KDF (such as defined in [RFC5869]) first, before using the key in a higher-level protocol.

9.4. Key confirmation

In many applications it is advisable to add an explicit key confirmation round after the CPace protocol flow. However, as some applications might only require implicit authentication and as explicit authentication messages are already a built-in feature in many higher-level protocols (e.g. TLS 1.3), the CPace protocol described here does not mandate key confirmation.

Already without explicit key confirmation, CPace enjoys weak forward security under the sCDH and sSDH assumptions [AHH21]. With added explicit confirmation, CPace enjoys perfect forward security also under the strong sCDH and sSDH assumptions [AHH21].

Note that in [ABKLX21] it was shown that an idealized variant of CPace also enjoys perfect forward security without explicit key confirmation. However this proof does not explicitly cover the recommended cipher suites in this document and requires the stronger assumption of an algebraic adversary model. For this reason, we recommend adding explicit key confirmation if perfect forward security is required.

When implementing explicit key confirmation, it is recommended to use an appropriate message-authentication code (MAC) such as HMAC [RFC2104] or CMAC [RFC4493] using a key mac_key derived from ISK.

One suitable option that works also in the parallel setting without message ordering is to proceed as follows.

* First calculate mac_key as mac_key = H.hash(b"CPaceMac" || sid || ISK).

* Then let each party send an authenticator tag calculated over the protocol message that it has sent previously. I.e. let party A calculate its authentication tag Ta as Ta = MAC(mac_key, lv_cat(Ya,ADa)) and let party B calculate its authentication tag Tb as Tb = MAC(mac_key, lv_cat(Yb,ADb)).

* Let the receiving party check the remote authentication tag for the correct value and abort in case that it’s incorrect.

9.5. Integrating CPace in higher-level protocols such as TLS1.3

When integrating CPace into a higher-level protocol such as TLS1.3 [RFC8446] it is recommended to use ISK as shared secret (which might otherwise be generated as part of a Diffie-Hellman key exchange output for other cipher suites).

Note that unlike the shared secret of a Diffie-Hellman protocol run, ISK will also provide mutual implicit authentication of the protocol partners. For providing explicit authentication, it is recommended to add a key confirmation round along the lines in Section 9.4, such as e.g. done in the "Finished" messages in TLS1.3 [RFC8446].

If an embedding protocol uses more than two messages (e.g. four message TLS1.3 [RFC8446] flows involving a hello-retry message and a repeated client-hello message) it is suggested that the CPace layer only considers the two messages used for the CPace run. I.e., it is suggested that authenticating the full message sequence involving also the additional messages that might precede the two CPace messages is done under the responsibility of the embedding application protocol. This could be done by integrating the full protocol transcript as part of a final explicit key confirmation round (as commonly done by TLS 1.3 as part of the "Finished" messages). Alternatively, information on communication rounds preceding the CPace flows can also be integrated as part of the CI field, as this will authenticate the information and will not require both communication partners to keep state information regarding preceding messages in memory until after the CPace run.

In case of TLS 1.3 [RFC8446] it is suggested to integrate Ya into the client-hello message and Yb into the server-hello message. Also party identifiers might best be added to the client-hello and server-hello messages as part of extension fields. It is recommended to use

the full octet stream encoding of the client-hello message as parameter ADa. Likewise it is recommended to use the encoding of the server-hello message for the parameter ADb. This approach has the drawback that the public points Ya and Yb might show up redundantly duplicated in the hashing operation for CPace's transcript strings but has the advantage of simplicity and the advantage that all meta-information in the extension fields within the client- and server hello fields will always become authenticated as part of the ISK.

9.6. Calculating a session identifier alongside with the CPace run

If CPace was run with an empty string sid available as input, both parties can produce a public session identifier string sid_output = H.hash(b"CPaceSidOutput" || transcript(Ya,ADa,Yb,ADb)) which will be unique for honest parties [BGHJ24].

9.7. Sampling of scalars

For curves over fields F_q where q is a prime close to a power of two, we recommend sampling scalars as a uniform bit string of length field_size_bits. We do so in order to reduce both, complexity of the implementation and the attack surface with respect to side-channels for embedded systems in hostile environments. [AHH21] demonstrated that non-uniform sampling did not negatively impact security for the case of Curve25519 and Curve448. This analysis however does not transfer to most curves in Short-Weierstrass form.

As a result, we recommend rejection sampling if G is as in Section 7.4. Alternatively an algorithm designed along the lines of the hash_to_field() function from [RFC9380] can also be used. There, oversampling to an integer significantly larger than the curve order is followed by a modular reduction to the group order.

9.8. Preconditions for using the simplified CPace specification from Section 7.2

The security of the algorithms used for the recommended cipher suites for the Montgomery curves Curve25519 and Curve448 in Section 7.2 rely on the following properties [AHH21]:

* The curve has order (p * c) with p prime and c a small cofactor. Also the curve's quadratic twist must be of order (p' * c') with p' prime and c' a cofactor.

* The cofactor c of the curve MUST BE equal to or an integer multiple of the cofactor c' of the curve's quadratic twist. Also, importantly, the implementation of the scalar_mult and scalar_mult_vfy functions must ensure that all scalars actually

used for the group operation are integer multiples of c (e.g. such as asserted by the specification of the decodeScalar functions in [RFC7748]).

* Both field order q and group order p MUST BE close to a power of two along the lines of [AHH21], Appendix E. Otherwise the simplified scalar sampling specified in Section 7.2 needs to be changed.

* The representation of the neutral element G.I MUST BE the same for both, the curve and its twist.

* The implementation of G.scalar_mult_vfy(y,X) MUST map all c low-order points on the curve and all c' low-order points on the twist to G.I.

Algorithms for curves other than the ones recommended here can be based on the principles from Section 7.2 given that the above properties hold.

9.9. Nonce values

Secret scalars ya and yb MUST NOT be reused. Values for sid SHOULD NOT be reused since the composability guarantees established by the simulation-based proof rely on the uniqueness of session ids [AHH21].

If the higher-level protocol that integrates CPace is able to establish a unique sid identifier for the communication session, it is RECOMMENDED that this is passed to CPace as sid parameter. One suitable option for generating sid is concatenation of ephemeral random strings contributed by both parties.

9.10. Side channel attacks

All state-of-the art methods for realizing constant-time execution SHOULD be used. Special care is RECOMMENDED specifically for elliptic curves in Short-Weierstrass form as important standard documents including [IEEE1363] describe curve operations with non-constant-time algorithms.

In case that side channel attacks are to be considered practical for a given application, it is RECOMMENDED to pay special attention on computing the secret generator G.calculate_generator(PRS,CI,sid). The most critical substep to consider might be the processing of the first block of the hash that includes the PRS string. The zero-padding introduced when hashing the sensitive PRS string can be expected to make the task for a side-channel attack somewhat more complex. Still this feature alone is not sufficient for ruling out

power analysis attacks. The mapping algorithm that converts the generator string to a elliptic curve point SHALL execute in constant time. In [RFC9380] suitable constant-time methods are available for any elliptic curve.

Even though the calculate_generator operation might be considered to form the primary target for side-channel attacks as information on long-term secrets might be exposed, also the subsequent operations on ephemeral values, such as scalar sampling and scalar multiplication should be protected from side-channels.

9.11. Large-characteristic finite fields

This document intentionally specifies CPace only for use on elliptic curve groups and the security proofs in [AHH21] only cover this case explicitly. For group environments built upon safe primes additional security analysis will be required. For instance exponential equivalence attacks may become practical when short exponents are used.

9.12. Quantum computers

CPace is proven secure under the hardness of the strong computational Simultaneous Diffie-Hellmann (sSDH) and strong computational Diffie-Hellmann (sCDH) assumptions in the group G (as defined in [AHH21]). These assumptions are not expected to hold any longer when large-scale quantum computers (LSQC) are available. Still, even in case that LSQC emerge, it is reasonable to assume that discrete-logarithm computations will remain costly. CPace with ephemeral pre-established session id values sid forces the adversary to solve one computational Diffie-Hellman problem per password guess [ES21]. If party identifiers are included as part of CI then the adversary is forced to solve one computational Diffie-Hellman problem per password guess and party identifier pair. For this reason it is RECOMMENDED to use the optional inputs sid if available in an application setting. For the same reason it is RECOMMENDED to integrate party identity strings A,B into CI.

In this sense, using the wording suggested by Steve Thomas on the CFRG mailing list, CPace is "quantum-annoying".

10. IANA Considerations

No IANA action is required.

11. Reference implementation and test vector generation

The reference implementation that was used for deriving test vectors is available at [REFIMP]. The embedded base64-encoded test vectors will decode to JSON files having the test vector’s octet strings encoded as base16 (i.e. hexadecimal) strings.

12. Acknowledgements

We would like to thank the participants on the CFRG list for comments and advice. Any comment and advice is appreciated.

13. References

13.1. Normative References

[I-D.draft-irtf-cfrg-ristretto255-decaf448] de Valence, H., Grigg, J., Hamburg, M., Lovecruft, I., Tankersley, G., and F. Valsorda, "The ristretto255 and decaf448 Groups", Work in Progress, Internet-Draft, draft-irtf-cfrg-ristretto255-decaf448-08, 5 September 2023, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-ristretto255-decaf448-08>.

[IEEE1363] "Standard Specifications for Public Key Cryptography, IEEE 1363", 2000.

[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/rfc/rfc2119>.

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

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

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

13.2. Informative References

[ABKLX21] Abdalla, M., Barbosa, M., Katz, J., Loss, J., and J. Xu, "Algebraic Adversaries in the Universal Composability Framework.", n.d., <https://eprint.iacr.org/2021/1218>.

[AHH21] Abdalla, M., Haase, B., and J. Hesse, "Security analysis of CPace", n.d., <https://eprint.iacr.org/2021/114>.

[BGHJ24] Barbosa, M., Gellert, K., Hesse, J., and S. Jarecki, "Bare PAKE: Universally Composable Key Exchange from Just Passwords", n.d., <link.springer.com/chapter/10.1007/978-3-031-68379-4_6>.

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

[ES21] Eaton, E. and D. Stebila, "The 'quantum annoying' property of password-authenticated key exchange protocols.", n.d., <https://eprint.iacr.org/2021/696>.

[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>.

[HMSD18] Hao, F., Metere, R., Sahahandashti, S., and C. Dong, "Analysing and Patching SPEKE in ISO/IEC", n.d., <https://arxiv.org/abs/1802.04900>.

[HS14] Hao, F. and S. Shahandashti, "The SPEKE Protocol Revisited", n.d., <https://eprint.iacr.org/2014/585.pdf>.

[I-D.irtf-cfrg-opaque] Bourdrez, D., Krawczyk, H., Lewi, K., and C. A. Wood, "The OPAQUE Augmented PAKE Protocol", Work in Progress, Internet-Draft, draft-irtf-cfrg-opaque-18, 21 November 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-18>.

[REFIMP] "CPace reference implementation (sage)", September 2024, <https://github.com/cfrg/draft-irtf-cfrg-cpace/tree/master/poc>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/rfc/rfc4493>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/rfc/rfc5246>.

[RFC5639] Lochter, M. and J. Merkle, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation", RFC 5639, DOI 10.17487/RFC5639, March 2010, <https://www.rfc-editor.org/rfc/rfc5639>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/rfc/rfc6234>.

[RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265, DOI 10.17487/RFC8265, October 2017, <https://www.rfc-editor.org/rfc/rfc8265>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC9380] Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R. S., and C. A. Wood, "Hashing to Elliptic Curves", RFC 9380, DOI 10.17487/RFC9380, August 2023, <https://www.rfc-editor.org/rfc/rfc9380>.

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

Appendix A. CPace function definitions

A.1. Definition and test vectors for string utility functions

A.1.1. prepend_len function

```
   def prepend_len(data):
       "prepend LEB128 encoding of length"
       length = len(data)
       length_encoded = b""
       while True:
           if length < 128:
               length_encoded += bytes([length])
           else:
               length_encoded += bytes([(length & 0x7f) + 0x80])
           length = int(length >> 7)
           if length == 0:
               break;
       return length_encoded + data
```

A.1.2. prepend_len test vectors

```
     prepend_len(b""): (length: 1 bytes)
       00
     prepend_len(b"1234"): (length: 5 bytes)
       0431323334
     prepend_len(bytes(range(127))): (length: 128 bytes)
       7f000102030405060708090a0b0c0d0e0f101112131415161718191a1b
       1c1d1e1f202122232425262728292a2b2c2d2e2f303132333435363738
       393a3b3c3d3e3f404142434445464748494a4b4c4d4e4f505152535455
       565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172
       737475767778797a7b7c7d7e
     prepend_len(bytes(range(128))): (length: 130 bytes)
       8001000102030405060708090a0b0c0d0e0f101112131415161718191a
       1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637
       38393a3b3c3d3e3f404142434445464748494a4b4c4d4e4f5051525354
       55565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f7071
       72737475767778797a7b7c7d7e7f
```

A.1.2.1. Testvectors as JSON file encoded as BASE64

```
    #eyJwcmVwZW5kX2xlbihiJycpIjogIjAwIiwgImInMTIzNCciOiAiMzEzMjMzMzQiLC
    #AicHJlcGVuZF9sZW4oYicxMjM0JykiOiAiMDQzMTMyMzMzNCIsICJwcmVwZW5kX2xl
    #bihieXRlcyhyYW5nZSgxMjcpKSkiOiAiN0YwMDAxMDIwMzA0MDUwNjA3MDgwOTBBME
    #IwQzBEMEUwRjEwMTExMjEzMTQxNTE2MTcxODE5MUExQjFDMUQxRTFGMjAyMTIyMjMy
    #NDI1MjYyNzI4MjkyQTJCMkMyRDJFMkYzMDMxMzIzMzM0MzUzNjM3MzgzOTNBM0IzQz
    #NEM0UzRjQwNDE0MjQzNDQ0NTQ2NDc0ODQ5NEE0QjRDNEQ0RTRGNTA1MTUyNTM1NDU1
    #NTY1NzU4NTk1QTVCNUM1RDVFNUY2MDYxNjI2MzY0NjU2NjY3Njg2OTZBNkI2QzZENk
    #U2RjcwNzE3MjczNzQ3NTc2Nzc3ODc5N0E3QjdDN0Q3RSIsICJwcmVwZW5kX2xlbihi
    #eXRlcyhyYW5nZSgxMjgpKSkiOiAiODAwMTAwMDEwMjAzMDQwNTA2MDcwODA5MEEwQj
    #BDMEQwRTBGMTAxMTEyMTMxNDE1MTYxNzE4MTkxQTFCMUMxRDFFMUYyMDIxMjIyMzI0
    #MjUyNjI3MjgyOTJBMkIyQzJEMkUyRjMwMzEzMjMzMzQzNTM2MzczODM5M0EzQjNDM0
    #QzRTNGNDA0MTQyNDM0NDQ1NDY0NzQ4NDk0QTRCNEM0RDRFNEY1MDUxNTI1MzU0NTU1
    #NjU3NTg1OTVBNUI1QzVENUU1RjYwNjE2MjYzNjQ2NTY2Njc2ODY5NkE2QjZDNkQ2RT
    #ZGNzA3MTcyNzM3NDc1NzY3Nzc4Nzk3QTdCN0M3RDdFN0YifQ==
```

A.1.3. lv_cat function

```
     def lv_cat(*args):
         result = b""
         for arg in args:
             result += prepend_len(arg)
         return result
```

A.1.4. Testvector for lv_cat()

```
     lv_cat(b"1234",b"5",b"",b"678"): (length: 12 bytes)
       043132333401350003363738
```

A.1.4.1. Testvectors as JSON file encoded as BASE64

```
    #eyJiJzEyMzQnIjogIjMxMzIzMzM0IiwgImInNSciOiAiMzUiLCAiYic2NzgnIjogIj
    #M2MzczOCIsICJsdl9jYXQoYicxMjM0JyxiJzUnLGInJyxiJzY3OCcpIjogIjA0MzEz
    #MjMzMzQwMTM1MDAwMzM2MzczOCJ9
```

A.2. Definition of generator_string function.

```
   def generator_string(DSI,PRS,CI,sid,s_in_bytes):
       # Concat all input fields with prepended length information.
       # Add zero padding in the first hash block after DSI and PRS.
       len_zpad = max(0,s_in_bytes - 1 - len(prepend_len(PRS))
                        - len(prepend_len(DSI)))
       return lv_cat(DSI, PRS, zero_bytes(len_zpad),
                              CI, sid)
```

A.3. Definitions and test vector ordered concatenation

A.3.1. Definitions for lexiographical ordering

For ordered concatenation lexiographical ordering of byte sequences is used:

```
def lexiographically_larger(bytes1,bytes2):
   "Returns True if bytes1>bytes2 using lexiographical ordering."
   min_len = min (len(bytes1), len(bytes2))
   for m in range(min_len):
       if bytes1[m] > bytes2[m]:
           return True;
       elif bytes1[m] < bytes2[m]:
           return False;
   return len(bytes1) > len(bytes2)
```

A.3.2. Definitions for ordered concatenation

With the above definition of lexiographical ordering ordered concatenation is specified as follows.

```
def o_cat(bytes1,bytes2):
    if lexiographically_larger(bytes1,bytes2):
        return b"oc" + bytes1 + bytes2
    else:
        return b"oc" + bytes2 + bytes1
```

A.3.3. Test vectors ordered concatenation

```
string comparison for o_cat:
  lexiographically_larger(b"\0", b"\0\0") == False
  lexiographically_larger(b"\1", b"\0\0") == True
  lexiographically_larger(b"\0\0", b"\0") == True
  lexiographically_larger(b"\0\0", b"\1") == False
  lexiographically_larger(b"\0\1", b"\1") == False
  lexiographically_larger(b"ABCD", b"BCD") == False

o_cat(b"ABCD",b"BCD"): (length: 9 bytes)
  6f6342434441424344
o_cat(b"BCD",b"ABCDE"): (length: 10 bytes)
  6f634243444142434445
```

A.3.3.1. Testvectors as JSON file encoded as BASE64

```
#eyJiJ0FCQ0QnIjogIjQxNDI0MzQ0IiwgImInQkNEJyI6ICI0MjQzNDQiLCAiYidBQk
#NERSciOiAiNDE0MjQzNDQ0NSIsICJvX2NhdChiJ0FCQ0QnLGInQkNEJykiOiAiNkY2
#MzQyNDM0NDQxNDI0MzQ0IiwgIm9fY2F0KGInQkNEJyxiJ0FCQ0RFJykiOiAiNkY2Mz
#QyNDM0NDQxNDI0MzQ0NDUifQ==
```

A.3.4. Definitions for transcript_ir function

```
   def transcript_ir(Ya,ADa,Yb,ADb):
       result = lv_cat(Ya,ADa) + lv_cat(Yb,ADb)
       return result
```

A.3.5. Test vectors transcript_ir function

```
     transcript_ir(b"123", b"PartyA", b"234",b"PartyB"):
     (length: 22 bytes)
       033132330650617274794103323334065061727479 42
     transcript_ir(b"3456",b"PartyA",b"2345",b"PartyB"):
     (length: 24 bytes)
       043334353606506172747941043233343506506172747942
```

A.3.5.1. Testvectors as JSON file encoded as BASE64

```
    #eyJiJzEyMyciOiAiMzEzMjMzIiwgImInMjM0JyI6ICIzMjMzMzQiLCAiYidQYXJ0eU
    #EnIjogIjUwNjE3Mjc0Nzk0MSIsICJiJ1BhcnR5QiciOiAiNTA2MTcyNzQ3OTQyIiwg
    #ImInMzQ1NiciOiAiMzMzNDM1MzYiLCAiYicyMzQ1JyI6ICIzMjMzMzQzNSIsICJ0cm
    #Fuc2NyaXB0X2lyKGInMTIzJyxiJ1BhcnR5QScsYicyMzQnLGInUGFydHlCJykiOiAi
    #MDMzMTMyMzMwNjUwNjE3Mjc0Nzk0MTAzMzIzMzM0MDY1MDYxNzI3NDc5NDIiLCAidH
    #JhbnNjcmlwdF9pcihiJzM0NTYnLGInUGFydHlBJyxiJzIzNDUnLGInUGFydHlCJyki
    #OiAiMDQzMzM0MzUzNjA2NTA2MTcyNzQ3OTQxMDQzMjMzMzQzNTA2NTA2MTcyNzQ3OT
    #QyIn0=
```

A.3.6. Definitions for transcript_oc function

```
   def transcript_oc(Ya,ADa,Yb,ADb):
       result = o_cat(lv_cat(Ya,ADa),lv_cat(Yb,ADb))
       return result
```

A.3.7. Test vectors for transcript_oc function

```
     transcript_oc(b"123", b"PartyA", b"234",b"PartyB"):
     (length: 24 bytes)
       6f6303323334065061727479420331323306506172747941
     transcript_oc(b"3456",b"PartyA",b"2345",b"PartyB"):
     (length: 26 bytes)
       6f6304333435360650617274794104323334350650617274 7942
```

A.3.7.1. Testvectors as JSON file encoded as BASE64

```
#eyJiJzEyMyciOiAiMzEzMjMzIiwgImInMjM0JyI6ICIzMjMzMzQiLCAiYidQYXJ0eU
#EnIjogIjUwNjE3Mjc0Nzk0MSIsICJiJ1BhcnR5QiciOiAiNTA2MTcyNzQ3OTQyIiwg
#ImInMzQ1NiciOiAiMzMzNDM1MzYiLCAiYicyMzQ1JyI6ICIzMjMzMzQzNSIsICJ0cm
#Fuc2NyaXB0X29jKGInMTIzJyxiJ1BhcnR5QScsYicyMzQnLGInUGFydHlCJykiOiAi
#NkY2MzAzMzIzMzM0MDY1MDYxNzI3NDc5NDIwMzMxMzIzMzA2NTA2MTcyNzQ3OTQxIi
#wgInRyYW5zY3JpcHRfb2MoYiczNDU2JyxiJ1BhcnR5QScsYicyMzQ1JyxiJ1BhcnR5
#QicpIjogIjZGNjMwNDMzMzQzNTM2MDY1MDYxNzI3NDc5NDEwNDMyMzMzNDM1MDY1MD
#YxNzI3NDc5NDIifQ==
```

A.4. Decoding and Encoding functions according to RFC7748

```
def decodeLittleEndian(b, bits):
    return sum([b[i] << 8*i for i in range((bits+7)/8)])

def decodeUCoordinate(u, bits):
    u_list = [ord(b) for b in u]
    # Ignore any unused bits.
    if bits % 8:
        u_list[-1] &= (1<<(bits%8))-1
    return decodeLittleEndian(u_list, bits)

def encodeUCoordinate(u, bits):
    return ''.join([chr((u >> 8*i) & 0xff)
                    for i in range((bits+7)/8)])
```

A.5. Elligator 2 reference implementation

The Elligator 2 map requires a non-square field element Z which shall be calculated as follows.

```
 def find_z_ell2(F):
     # Find nonsquare for Elligator2
     # Argument: F, a field object, e.g., F = GF(2^255 - 19)
     ctr = F.gen()
     while True:
         for Z_cand in (F(ctr), F(-ctr)):
             # Z must be a non-square in F.
             if is_square(Z_cand):
                 continue
             return Z_cand
         ctr += 1
```

The values of the non-square Z only depend on the curve. The algorithm above results in a value of Z = 2 for Curve25519 and Z=-1 for Ed448.

The following code maps a field element r to an encoded field element which is a valid u-coordinate of a Montgomery curve with curve parameter A.

```
    def elligator2(r, q, A, field_size_bits):
        # Inputs: field element r, field order q,
        #         curve parameter A and field size in bits
        Fq = GF(q); A = Fq(A); B = Fq(1);

        # get non-square z as specified in the hash2curve draft.
        z = Fq(find_z_ell2(Fq))
        powerForLegendreSymbol = floor((q-1)/2)

        v = - A / (1 + z * r^2)
        epsilon = (v^3 + A * v^2 + B * v)^powerForLegendreSymbol
        x = epsilon * v - (1 - epsilon) * A/2
        return encodeUCoordinate(Integer(x), field_size_bits)
```

Appendix B. Test vectors

B.1. Test vector for CPace using group X25519 and hash SHA-512

B.1.1. Test vectors for calculate_generator with group X25519

```
     Inputs
       H   = SHA-512 with input block size 128 bytes.
       PRS = b'Password' ; ZPAD length: 109 ; DSI = b'CPace255'
       CI = b'oc\x0bB_responder\x0bA_initiator'
       CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
       sid = 7e4b4791d6a8ef019b936c79fb7f2c57
     Outputs
       generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
       (length: 172 bytes)
         0843506163653235350850617373776f72646d00000000000000000
         0000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000
         0000000000000000000000000001a6f630b425f726573706f6e
         6465720b415f696e69746961746f72107e4b4791d6a8ef019b936c79
         fb7f2c57
       hash generator string: (length: 32 bytes)
         92806dc608984dbf4e4aae478c6ec453ae979cc01ecc1a2a7cf49f5c
         ee56551b
       decoded field element of 255 bits: (length: 32 bytes)
         92806dc608984dbf4e4aae478c6ec453ae979cc01ecc1a2a7cf49f5c
         ee56551b
       generator g: (length: 32 bytes)
         64e8099e3ea682cfdc5cb665c057ebb514d06bf23ebc9f743b51b822
         42327074
```

B.1.1.1. Testvectors as JSON file encoded as BASE64

```
    #eyJIIjogIlNIQS01MTIiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
    #E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiAxMDksICJEU0kiOiAiNDM1MDYx
    #NjM2NTMyMzUzNSIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NTczNzA2RjZFNjQ2NTcyME
    #I0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI3RTRCNDc5MUQ2QThFRjAx
    #OUI5MzZDNzlGQjdGMkM1NyIsICJnZW5lcmF0b3Jfc3RyaW5nKEcuRFNJLFBSUyxDSS
    #xzaWQsSC5zX2luX2J5dGVzKSI6ICIwODQzNTA2MTYzNjUzMjM1MzUwODUwNjE3Mzcz
    #Nzc2RjcyNjQ2RDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
    #MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
    #MDAwMDAwMDAwMDAwMDAwMDAwMDAwMUE2RjYzMEI0MjVGNzI2NTczNz
    #A2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyMTA3RTRCNDc5MUQ2QThF
    #RjAxOUI5MzZDNzlGQjdGMkM1NyIsICJoYXNoIGdlbmVyYXRvciBzdHJpbmciOiAiOT
    #I4MDZEQzYwODk4NERCRjRFNEFBRTQ3OEM2RUM0NTNBRTk3OUNDMDFFQ0MxQTJBN0NG
    #NDlGNUNFRTU2NTUxQiIsICJkZWNvZGVkIGZpZWxkIGVsZW1lbnQgb2YgMjU1IGJpdH
    #MiOiAiOTI4MDZEQzYwODk4NERCRjRFNEFBRTQ3OEM2RUM0NTNBRTk3OUNDMDFFQ0Mx
    #QTJBN0NGNDlGNUNFRTU2NTUxQiIsICJnZW5lcmF0b3IgZyI6ICI2NEU4MDk5RTNFQT
    #Y4MkNGREM1Q0I2NjVDMDU3RUJCNTE0RDA2QkYyM0VCQzlGNzQzQjUxQjgyMjQyMzI3
    #MDc0In0=
```

B.1.2. Test vector for message from A

```
Inputs
  ADa = b'ADa'
  ya (little endian): (length: 32 bytes)
    21b4f4bd9e64ed355c3eb676a28ebedaf6d8f17bdc365995b3190971
    53044080
Outputs
  Ya: (length: 32 bytes)
    1b02dad6dbd29a07b6d28c9e04cb2f184f0734350e32bb7e62ff9dbc
    fdb63d15
```

B.1.3. Test vector for message from B

```
Inputs
  ADb = b'ADb'
  yb (little endian): (length: 32 bytes)
    848b0779ff415f0af4ea14df9dd1d3c29ac41d836c7808896c4eba19
    c51ac40a
Outputs
  Yb: (length: 32 bytes)
    20cda5955f82c4931545bcbf40758ce1010d7db4db2a907013d79c7a
    8fcf957f
```

B.1.4. Test vector for secret points K

```
  scalar_mult_vfy(ya,Yb): (length: 32 bytes)
    f97fdfcfff1c983ed6283856a401de3191ca919902b323c5f950c970
    3df7297a
  scalar_mult_vfy(yb,Ya): (length: 32 bytes)
    f97fdfcfff1c983ed6283856a401de3191ca919902b323c5f950c970
    3df7297a
```

B.1.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 74 bytes)
  201b02dad6dbd29a07b6d28c9e04cb2f184f0734350e32bb7e62ff9d
  bcfdb63d15034144612020cda5955f82c4931545bcbf40758ce1010d
  7db4db2a907013d79c7a8fcf957f03414462
DSI = G.DSI_ISK, b'CPace255_ISK': (length: 12 bytes)
  435061636532353 55f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 137 bytes)
  0c435061636532353 55f49534b107e4b4791d6a8ef019b936c79fb7f
  2c5720f97fdfcfff1c983ed6283856a401de3191ca919902b323c5f9
  50c9703df7297a201b02dad6dbd29a07b6d28c9e04cb2f184f073435
  0e32bb7e62ff9dbcfdb63d15034144612020cda5955f82c4931545bc
  bf40758ce1010d7db4db2a907013d79c7a8fcf957f03414462
ISK result: (length: 64 bytes)
  a051ee5ee2499d16da3f69f430218b8ea94a18a45b67f9e86495b382
  c33d14a5c38cecc0cc834f960e39e0d1bf7d76b9ef5d54eecc5e0f38
  6c97ad12da8c3d5f
```

B.1.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 76 bytes)
  6f632020cda5955f82c4931545bcbf40758ce1010d7db4db2a907013
  d79c7a8fcf957f03414462201b02dad6dbd29a07b6d28c9e04cb2f18
  4f0734350e32bb7e62ff9dbcfdb63d1503414461
DSI = G.DSI_ISK, b'CPace255_ISK': (length: 12 bytes)
  435061636532353 55f49534b
lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
(length: 139 bytes)
  0c435061636532353 55f49534b107e4b4791d6a8ef019b936c79fb7f
  2c5720f97fdfcfff1c983ed6283856a401de3191ca919902b323c5f9
  50c9703df7297a6f632020cda5955f82c4931545bcbf40758ce1010d
  7db4db2a907013d79c7a8fcf957f03414462201b02dad6dbd29a07b6
  d28c9e04cb2f184f0734350e32bb7e62ff9dbcfdb63d1503414461
ISK result: (length: 64 bytes)
  5cc27e49679423f81a37d7521d9fb1327c840d2ea4a1543652e7de5c
  abb89ebad27d24761b3288a3fd5764b441ecb78d30abc26161ff45ea
  297bb311dde04727
```

B.1.7. Test vector for optional output of session id

```
      H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        f7ae11ac3ee85c3c42d8bd51ba823fbe17158f43d34a1296f1cb2567
        bcc71dc8b201a134b566b468aad8fd04f02f96e3caf9d5601f7ed760
        a0a951a5a861b5e7
      H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        a38389e34fa492788c1df43b06b427710491174e53c33b01362a490d
        116fe1b7e870aa6e2a7fc018725e3b7f969f7508042e44cd3863f39a
        a75026a190d1902b
```

B.1.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
    0x7e,0x4b,0x47,0x91,0xd6,0xa8,0xef,0x01,0x9b,0x93,0x6c,0x79,
    0xfb,0x7f,0x2c,0x57,
   };
   const unsigned char tc_g[] = {
    0x64,0xe8,0x09,0x9e,0x3e,0xa6,0x82,0xcf,0xdc,0x5c,0xb6,0x65,
    0xc0,0x57,0xeb,0xb5,0x14,0xd0,0x6b,0xf2,0x3e,0xbc,0x9f,0x74,
    0x3b,0x51,0xb8,0x22,0x42,0x32,0x70,0x74,
   };
   const unsigned char tc_ya[] = {
    0x21,0xb4,0xf4,0xbd,0x9e,0x64,0xed,0x35,0x5c,0x3e,0xb6,0x76,
    0xa2,0x8e,0xbe,0xda,0xf6,0xd8,0xf1,0x7b,0xdc,0x36,0x59,0x95,
    0xb3,0x19,0x09,0x71,0x53,0x04,0x40,0x80,
   };
   const unsigned char tc_ADa[] = {
    0x41,0x44,0x61,
   };
   const unsigned char tc_Ya[] = {
    0x1b,0x02,0xda,0xd6,0xdb,0xd2,0x9a,0x07,0xb6,0xd2,0x8c,0x9e,
    0x04,0xcb,0x2f,0x18,0x4f,0x07,0x34,0x35,0x0e,0x32,0xbb,0x7e,
    0x62,0xff,0x9d,0xbc,0xfd,0xb6,0x3d,0x15,
   };
   const unsigned char tc_yb[] = {
    0x84,0x8b,0x07,0x79,0xff,0x41,0x5f,0x0a,0xf4,0xea,0x14,0xdf,
    0x9d,0xd1,0xd3,0xc2,0x9a,0xc4,0x1d,0x83,0x6c,0x78,0x08,0x89,
    0x6c,0x4e,0xba,0x19,0xc5,0x1a,0xc4,0x0a,
   };
```

```
   const unsigned char tc_ADb[] = {
    0x41,0x44,0x62,
   };
   const unsigned char tc_Yb[] = {
    0x20,0xcd,0xa5,0x95,0x5f,0x82,0xc4,0x93,0x15,0x45,0xbc,0xbf,
    0x40,0x75,0x8c,0xe1,0x01,0x0d,0x7d,0xb4,0xdb,0x2a,0x90,0x70,
    0x13,0xd7,0x9c,0x7a,0x8f,0xcf,0x95,0x7f,
   };
   const unsigned char tc_K[] = {
    0xf9,0x7f,0xdf,0xcf,0xff,0x1c,0x98,0x3e,0xd6,0x28,0x38,0x56,
    0xa4,0x01,0xde,0x31,0x91,0xca,0x91,0x99,0x02,0xb3,0x23,0xc5,
    0xf9,0x50,0xc9,0x70,0x3d,0xf7,0x29,0x7a,
   };
   const unsigned char tc_ISK_IR[] = {
    0xa0,0x51,0xee,0x5e,0xe2,0x49,0x9d,0x16,0xda,0x3f,0x69,0xf4,
    0x30,0x21,0x8b,0x8e,0xa9,0x4a,0x18,0xa4,0x5b,0x67,0xf9,0xe8,
    0x64,0x95,0xb3,0x82,0xc3,0x3d,0x14,0xa5,0xc3,0x8c,0xec,0xc0,
    0xcc,0x83,0x4f,0x96,0x0e,0x39,0xe0,0xd1,0xbf,0x7d,0x76,0xb9,
    0xef,0x5d,0x54,0xee,0xcc,0x5e,0x0f,0x38,0x6c,0x97,0xad,0x12,
    0xda,0x8c,0x3d,0x5f,
   };
   const unsigned char tc_ISK_SY[] = {
    0x5c,0xc2,0x7e,0x49,0x67,0x94,0x23,0xf8,0x1a,0x37,0xd7,0x52,
    0x1d,0x9f,0xb1,0x32,0x7c,0x84,0x0d,0x2e,0xa4,0xa1,0x54,0x36,
    0x52,0xe7,0xde,0x5c,0xab,0xb8,0x9e,0xba,0xd2,0x7d,0x24,0x76,
    0x1b,0x32,0x88,0xa3,0xfd,0x57,0x64,0xb4,0x41,0xec,0xb7,0x8d,
    0x30,0xab,0xc2,0x61,0x61,0xff,0x45,0xea,0x29,0x7b,0xb3,0x11,
    0xdd,0xe0,0x47,0x27,
   };
   const unsigned char tc_ISK_SY[] = {
    0x5c,0xc2,0x7e,0x49,0x67,0x94,0x23,0xf8,0x1a,0x37,0xd7,0x52,
    0x1d,0x9f,0xb1,0x32,0x7c,0x84,0x0d,0x2e,0xa4,0xa1,0x54,0x36,
    0x52,0xe7,0xde,0x5c,0xab,0xb8,0x9e,0xba,0xd2,0x7d,0x24,0x76,
    0x1b,0x32,0x88,0xa3,0xfd,0x57,0x64,0xb4,0x41,0xec,0xb7,0x8d,
    0x30,0xab,0xc2,0x61,0x61,0xff,0x45,0xea,0x29,0x7b,0xb3,0x11,
    0xdd,0xe0,0x47,0x27,
   };
   const unsigned char tc_sid_out_ir[] = {
    0xf7,0xae,0x11,0xac,0x3e,0xe8,0x5c,0x3c,0x42,0xd8,0xbd,0x51,
    0xba,0x82,0x3f,0xbe,0x17,0x15,0x8f,0x43,0xd3,0x4a,0x12,0x96,
    0xf1,0xcb,0x25,0x67,0xbc,0xc7,0x1d,0xc8,0xb2,0x01,0xa1,0x34,
    0xb5,0x66,0xb4,0x68,0xaa,0xd8,0xfd,0x04,0xf0,0x2f,0x96,0xe3,
    0xca,0xf9,0xd5,0x60,0x1f,0x7e,0xd7,0x60,0xa0,0xa9,0x51,0xa5,
    0xa8,0x61,0xb5,0xe7,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xa3,0x83,0x89,0xe3,0x4f,0xa4,0x92,0x78,0x8c,0x1d,0xf4,0x3b,
    0x06,0xb4,0x27,0x71,0x04,0x91,0x17,0x4e,0x53,0xc3,0x3b,0x01,
```

```
    0x36,0x2a,0x49,0x0d,0x11,0x6f,0xe1,0xb7,0xe8,0x70,0xaa,0x6e,
    0x2a,0x7f,0xc0,0x18,0x72,0x5e,0x3b,0x7f,0x96,0x9f,0x75,0x08,
    0x04,0x2e,0x44,0xcd,0x38,0x63,0xf3,0x9a,0xa7,0x50,0x26,0xa1,
    0x90,0xd1,0x90,0x2b,
   };
```

B.1.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
    #czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI3
    #RTRCNDc5MUQ2QThFRjAxOUI5MzZDNzlGQjdGMkM1NyIsICJnIjogIjY0RTgwOTlFM0
    #VBNjgyQ0ZEQzVDQjY2NUMwNTdFQkI1MTREMDZCRjIzRUJDOUY3NDNCNTFCODIyNDIz
    #MjcwNzQiLCAieWEiOiAiMjFCNEY0QkQ5RTY0RUQzNTVDM0VCNjc2QTI4RUJFREFGNk
    #Q4RjE3QkRDMzY1OTk1QjMxOTA5NzE1MzA0NDA4MCIsICJBRGEiOiAiNDE0NDYxIiwg
    #IllhIjogIjFCMDJEQUQ2REJEMjlBMDdCNkQyOEM5RTA0Q0IyRjE4NEYwNzM0MzUwRT
    #MyQkI3RTYyRkY5REJDRkRCNjNEMTUiLCAieWIiOiAiODQ4QjA3NzlGRjQxNUYwQUY0
    #RUExNERGOUREMUQzQzI5QUM0MUQ4MzZDNzgwODg5NkM0RUJBMTlDNTFBQzQwQSIsIC
    #JBRGIiOiAiNDE0NDYyIiwgIlliIjogIjIwQ0RBNTk1NUY4MkM0OTMxNTQ1QkNCRjQw
    #NzU4Q0UxMDEwRDdEQjREQjJBOTA3MDEzRDc5QzdBOEZDRjk1N0YiLCAiSyI6ICJGOT
    #dGREZDRkZGMUM5ODNFRDYyODM4NTZBNDAxREUzMTkxQ0E5MTk5MDJCMzIzQzVGOTUw
    #Qzk3MDNERjcyOTdBIiwgIklTS19JUiI6ICJBMDUxRUU1RUUyNDk5RDE2REEzRjY5Rj
    #QzMDIxOEI4RUE5NEExOEE0NUI2N0Y5RTg2NDk1QjM4MkMzM0QxNEE1QzM4Q0VDQzBD
    #QzgzNEY5NjBFMzlFMEQxQkY3RDc2QjlFRjVENTRFRUNDNUUwRjM4NkM5N0FEMTJEQT
    #hDM0Q1RiIsICJJU0tfU1kiOiAiNUNDMjdFNDk2Nzk0MjNGODFBMzdENzUyMUQ5RkIx
    #MzI3Qzg0MEQyRUE0QTE1NDM2NTJFN0RFNUNBQkI4OUVCQUQyN0QyNDc2MUIzMjg4QT
    #NGRDU3NjRCNDQxRUNCNzhEMzBBQkMyNjE2MUZGNDVFQTI5N0JCMzExRERFMDQ3Mjci
    #LCAic2lkX291dHB1dF9pciI6ICJGN0FFMTFBQzNFRTg1QzNDNDJEOEJENTFCQTgyM0
    #ZCRTE3MTU4RjQzRDM0QTEyOTZGMUNCMjU2N0JDQzcxREM4QjIwMUExMzRCNTY2QjQ2
    #OEFBRDhGRDA0RjAyRjk2RTNDQUY5RDU2MDFGN0VENzYwQTBBOTUxQTVBODYxQjVFNy
    #IsICJzaWRfb3V0cHV0X29jIjogIkEzODM4OUUzNEZBNDkyNzg4QzFERjQzQjA2QjQy
    #NzcxMDQ5MTE3NEU1M0MzM0IwMTM2MkE0OTBEMTE2RkUxQjdFODcwQUE2RTJBN0ZDMD
    #E4NzI1RTNCN0Y5NjlGNzUwODA0MkU0NENEMzg2M0YzOUFBNzUwMjZBMTkwRDE5MDJC
    #In0=
```

B.1.10. Test vectors for G_X25519.scalar_mult_vfy: low order points

Test vectors for which G_X25519.scalar_mult_vfy(s_in,ux) must return the neutral element or would return the neutral element if bit #255 of field element representation was not correctly cleared. (The decodeUCoordinate function from RFC7748 mandates clearing bit #255 for field element representations for use in the X25519 function.).

```
   u0: 0000000000000000000000000000000000000000000000000000000000000000
   u1: 0100000000000000000000000000000000000000000000000000000000000000
   u2: ecffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff7f
   u3: e0eb7a7c3b41b8ae1656e3faf19fc46ada098deb9c32b1fd866205165f49b800
   u4: 5f9c95bca3508c24b1d0b1559c83ef5b04445cc4581c8e86d8224eddd09f1157
   u5: edffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff7f
   u6: daffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   u7: eeffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff7f
   u8: dbffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   u9: d9ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
   ua: cdeb7a7c3b41b8ae1656e3faf19fc46ada098deb9c32b1fd866205165f49b880
   ub: 4c9c95bca3508c24b1d0b1559c83ef5b04445cc4581c8e86d8224eddd09f11d7

   u0 ... ub MUST be verified to produce the correct results q0 ... qb:

   Additionally, u0,u1,u2,u3,u4,u5 and u7 MUST trigger the abort case
   when included in message from A or B.

   s = af46e36bf0527c9d3b16154b82465edd62144c0ac1fc5a18506a2244ba449aff
   qN = G_X25519.scalar_mult_vfy(s, uX)
   q0: 0000000000000000000000000000000000000000000000000000000000000000
   q1: 0000000000000000000000000000000000000000000000000000000000000000
   q2: 0000000000000000000000000000000000000000000000000000000000000000
   q3: 0000000000000000000000000000000000000000000000000000000000000000
   q4: 0000000000000000000000000000000000000000000000000000000000000000
   q5: 0000000000000000000000000000000000000000000000000000000000000000
   q6: d8e2c776bbacd510d09fd9278b7edcd25fc5ae9adfba3b6e040e8d3b71b21806
   q7: 0000000000000000000000000000000000000000000000000000000000000000
   q8: c85c655ebe8be44ba9c0ffde69f2fe10194458d137f09bbff725ce58803cdb38
   q9: db64dafa9b8fdd136914e61461935fe92aa372cb056314e1231bc4ec12417456
   qa: e062dcd5376d58297be2618c7498f55baa07d7e03184e8aada20bca28888bf7a
   qb: 993c6ad11c4c29da9a56f7691fd0ff8d732e49de6250b6c2e80003ff4629a175
```

B.1.10.1. Testvectors as JSON file encoded as BASE64

```
#eyJJbnZhbGlkIFkwIjogIjAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAiLCAiSW52YWxpZCBZMSI6ICIw
#MTAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwIiwgIkludmFsaWQgWTIiOiAiRUNGRkZGRkZGRkZGRkZG
#RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkY3RiIsIC
#JJbnZhbGlkIFkzIjogIkUwRUI3QTdDM0I0MUI4QUUxNjU2RTNGQUYxOUZDNDZBREEw
#OThERUI5QzMyQjFGRDg2NjIwNTE2NUY0OUI4MDAiLCAiSW52YWxpZCBZNCI6ICI1Rj
#lDOTVCQ0EzNTA4QzI0QjFEMEIxNTU5QzgzRUY1QjA0NDQ1Q0M0NTgxQzhFODZEODIy
#NEVEREQwOUYxMTU3IiwgIkludmFsaWQgWTUiOiAiRURGRkZGRkZGRkZGRkZGRk
#ZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkY3RiIsICJJ
#bnZhbGlkIFk2IjogIkRBRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRk
#ZGRkZGRkZGRkZGRkZGRkZGRkZGRkYiLCAiSW52YWxpZCBZNyI6ICJFRUZG
#RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRk
#ZGRkZGRjdGIiwgIkludmFsaWQgWTgiOiAiREJGRkZGRkZGRkZGRkZGRkZG
#RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRiIsICJJbn
#ZhbGlkIFk5IjogIkQ5RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZG
#RkZGRkZGRkZGRkZGRkYiLCAiSW52YWxpZCBZMTAiOiAiQ0RFQj
#dBN0MzQjQxQjhBRTE2NTZFM0ZBRjE5RkM0NkFEQTA5OERFQjlDMzJCMUZEODY2MjA1
#MTY1RjQ5Qjg4MCIsICJJbnZhbGlkIFkxMSI6ICI0QzlDOTVCQ0EzNTA4QzI0QjFEME
#IxNTU5QzgzRUY1QjA0NDQ1Q0M0NTgxQzhFODZEODIyNEVEREQwOUYxMUQ3In0=
```

B.2. Test vector for CPace using group X448 and hash SHAKE-256

B.2.1. Test vectors for calculate_generator with group X448

```
     Inputs
       H   = SHAKE-256 with input block size 136 bytes.
       PRS = b'Password' ; ZPAD length: 117 ; DSI = b'CPace448'
       CI = b'oc\x0bB_responder\x0bA_initiator'
       CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
       sid = 5223e0cdc45d6575668d64c552004124
     Outputs
       generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
       (length: 180 bytes)
         0843506163653434380850617373776f726475000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000001a6f630b
         425f726573706f6e6465720b415f696e69746961746f72105223e0cd
         c45d6575668d64c552004124
       hash generator string: (length: 56 bytes)
         98b713f84529194d719a7d86cb0f504b8afeb05354d68747e18b2e7c
         c8b6da526085e4263bd8bea7d69e479ebad09e30ae062e5d089da7f3
       decoded field element of 448 bits: (length: 56 bytes)
         98b713f84529194d719a7d86cb0f504b8afeb05354d68747e18b2e7c
         c8b6da526085e4263bd8bea7d69e479ebad09e30ae062e5d089da7f3
       generator g: (length: 56 bytes)
         e293b7ccf61ca7eb928a26391cf38b660f874a001fdf0bf3a91fd182
         f2b6d83e61a9377ede127eba7e0d4c08592eaff33d4aa705d6ce54bb
```

B.2.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQUtFLTI1NiIsICJILnNfaW5fYnl0ZXMiOiAxMzYsICJQUlMiOiAiNT
#A2MTczNzM3NzZGNzI2NCIsICJaUEFEIGxlbmd0aCI6IDExNywgIkRTSSI6ICI0MzUw
#NjE2MzY1MzQzNDM4IiwgIkNJIjogIjZGNjMwQjQyNUY3MjY1NzM3MDZGNkU2NDY1Nz
#IwQjQxNUY2OTZFNjk3NDY5NjE3NDZGNzIiLCAic2lkIjogIjUyMjNFMENEQzQ1RDY1
#NzU2NjhENjRDNTUyMDA0MTI0IiwgImdlbmVyYXRvcl9zdHJpbmcoRy5EU0ksUFJTLE
#NJLHNpZCxILnNfaW5fYnl0ZXMpIjogIjA4NDM1MDYxNjM2NTM0MzQzODA4NTA2MTcz
#NzM3NzZGNzI2NDc1MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMU
#E2RjYzMEI0MjVGNzI2NTczNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2
#RjcyMTA1MjIzRTBDREM0NUQ2NTc1NjY4RDY0QzU1MjAwNDEyNCIsICJoYXNoIGdlbm
#VyYXRvciBzdHJpbmciOiAiOThCNzEzRjg0NTI5MTk0RDcxOUE3RDg2Q0IwRjUwNEI4
#QUZFQjA1MzU0RDY4NzQ3RTE4QjJFN0NDOEI2REE1MjYwODVFNDI2M0JEOEJFQTdENj
#lFNDc5RUJBRDA5RTMwQUUwNjJFNUQwODlEQTdGMyIsICJkZWNvZGVkIGZpZWxkIGVs
#ZW1lbnQgb2YgNDQ4IGJpdHMiOiAiOThCNzEzRjg0NTI5MTk0RDcxOUE3RDg2Q0IwRj
#UwNEI4QUZFQjA1MzU0RDY4NzQ3RTE4QjJFN0NDOEI2REE1MjYwODVFNDI2M0JEOEJF
#QTdENjlFNDc5RUJBRDA5RTMwQUUwNjJFNUQwODlEQTdGMyIsICJnZW5lcmF0b3IgZy
#I6ICJFMjkzQjdDQ0Y2MUNBN0VCOTI4QTI2MzkxQ0YzOEI2NjBGODc0QTAwMUZERjBC
#RjNBOTFGRDE4MkYyQjZEODNFNjFBOTM3N0VERTEyN0VCQTdFMEQ0QzA4NTkyRUFGRj
#MzRDRBQTcwNUQ2Q0U1NEJCIn0=
```

B.2.2. Test vector for message from A

```
Inputs
  ADa = b’ADa’
  ya (little endian): (length: 56 bytes)
    21b4f4bd9e64ed355c3eb676a28ebedaf6d8f17bdc365995b3190971
    53044080516bd083bfcce66121a3072646994c8430cc382b8dc543e8
Outputs
  Ya: (length: 56 bytes)
    7f645772cc209bf9fd9d76dbb10283bea71b12235e3bb21878d5e56a
    70506e165743a632de98eca9932c5d2efe36500a59b2fdaed0d8a148
```

B.2.3. Test vector for message from B

```
Inputs
  ADb = b’ADb’
  yb (little endian): (length: 56 bytes)
    848b0779ff415f0af4ea14df9dd1d3c29ac41d836c7808896c4eba19
    c51ac40a439caf5e61ec88c307c7d619195229412eaa73fb2a5ea20d
Outputs
  Yb: (length: 56 bytes)
    a4690a0750c42b288ddd0ba08e3f4902dfe70bae5c9e2c6ee95844de
    f2692be77646b20d3b429f8da00d21433ee0891c667658d8d0c48e38
```

B.2.4. Test vector for secret points K

```
scalar_mult_vfy(ya,Yb): (length: 56 bytes)
  db3fff9da59576715b04d4df8dc8d18db2430e57bbed337dbeee5bb2
  d6ab6ceddc9c75c5c0b17fad7eb724daa12f8f1903dd6c2cede6135a
scalar_mult_vfy(yb,Ya): (length: 56 bytes)
  db3fff9da59576715b04d4df8dc8d18db2430e57bbed337dbeee5bb2
  d6ab6ceddc9c75c5c0b17fad7eb724daa12f8f1903dd6c2cede6135a
```

B.2.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 122 bytes)
  387f645772cc209bf9fd9d76dbb10283bea71b12235e3bb21878d5e5
  6a70506e165743a632de98eca9932c5d2efe36500a59b2fdaed0d8a1
  480341446138a4690a0750c42b288ddd0ba08e3f4902dfe70bae5c9e
  2c6ee95844def2692be77646b20d3b429f8da00d21433ee0891c6676
  58d8d0c48e3803414462
DSI = G.DSI_ISK, b'CPace448_ISK': (length: 12 bytes)
  435061636534343385f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 209 bytes)
  0c435061636534343385f49534b105223e0cdc45d6575668d64c55200
  412438db3fff9da59576715b04d4df8dc8d18db2430e57bbed337dbe
  ee5bb2d6ab6ceddc9c75c5c0b17fad7eb724daa12f8f1903dd6c2ced
  e6135a387f645772cc209bf9fd9d76dbb10283bea71b12235e3bb218
  78d5e56a70506e165743a632de98eca9932c5d2efe36500a59b2fdae
  d0d8a1480341446138a4690a0750c42b288ddd0ba08e3f4902dfe70b
  ae5c9e2c6ee95844def2692be77646b20d3b429f8da00d21433ee089
  1c667658d8d0c48e3803414462
ISK result: (length: 64 bytes)
  599892a2078a8c988181625e1e5e5f7a6163f7d72f21b93ebefba0f1
  7ff7ea3aa0594bd569cf74264157b3c0087bdccf2f59c77156628487
  f5ca1645b8e9d05b
```

B.2.6. Test vector for ISK calculation parallel execution

```
       transcript_oc(Ya,ADa,Yb,ADb): (length: 124 bytes)
         6f6338a4690a0750c42b288ddd0ba08e3f4902dfe70bae5c9e2c6ee9
         5844def2692be77646b20d3b429f8da00d21433ee0891c667658d8d0
         c48e3803414462387f645772cc209bf9fd9d76dbb10283bea71b1223
         5e3bb21878d5e56a70506e165743a632de98eca9932c5d2efe36500a
         59b2fdaed0d8a14803414461
       DSI = G.DSI_ISK, b'CPace448_ISK': (length: 12 bytes)
         435061636534343 85f49534b
       lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
       (length: 211 bytes)
         0c435061636534343 85f49534b105223e0cdc45d6575668d64c55200
         412438db3fff9da59576715b04d4df8dc8d18db2430e57bbed337dbe
         ee5bb2d6ab6ceddc9c75c5c0b17fad7eb724daa12f8f1903dd6c2ced
         e6135a6f6338a4690a0750c42b288ddd0ba08e3f4902dfe70bae5c9e
         2c6ee95844def2692be77646b20d3b429f8da00d21433ee0891c6676
         58d8d0c48e3803414462387f645772cc209bf9fd9d76dbb10283bea7
         1b12235e3bb21878d5e56a70506e165743a632de98eca9932c5d2efe
         36500a59b2fdaed0d8a14803414461
       ISK result: (length: 64 bytes)
         3ac73f03030296aa591f01326b18afa47e1189129cd06ae8dfb05e6e
         b1310cde948b59eef0755365c06a339266afe594948c56a538d98a65
         767113938a9a78d8
```

B.2.7. Test vector for optional output of session id

```
       H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         00a2333a79481abe71efd6594d7bbaac55c808482e869c9b65c4b53d
         7100d3da8f3cabd59fa0c1f22d6d2f9ac0c093962292798fca2c0b93
         268974cad75d575a
       H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         a1ce90537a8d53b06d77e79fe719461cc5ed8300d21d1866a59f9638
         601833f57a8b5e88db9a52abfa1b4e8a651a400bc9205082aad81eb3
         11c44373b9a19eff
```

B.2.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
    0x52,0x23,0xe0,0xcd,0xc4,0x5d,0x65,0x75,0x66,0x8d,0x64,0xc5,
```

```
 0x52,0x00,0x41,0x24,
};
const unsigned char tc_g[] = {
 0xe2,0x93,0xb7,0xcc,0xf6,0x1c,0xa7,0xeb,0x92,0x8a,0x26,0x39,
 0x1c,0xf3,0x8b,0x66,0x0f,0x87,0x4a,0x00,0x1f,0xdf,0x0b,0xf3,
 0xa9,0x1f,0xd1,0x82,0xf2,0xb6,0xd8,0x3e,0x61,0xa9,0x37,0x7e,
 0xde,0x12,0x7e,0xba,0x7e,0x0d,0x4c,0x08,0x59,0x2e,0xaf,0xf3,
 0x3d,0x4a,0xa7,0x05,0xd6,0xce,0x54,0xbb,
};
const unsigned char tc_ya[] = {
 0x21,0xb4,0xf4,0xbd,0x9e,0x64,0xed,0x35,0x5c,0x3e,0xb6,0x76,
 0xa2,0x8e,0xbe,0xda,0xf6,0xd8,0xf1,0x7b,0xdc,0x36,0x59,0x95,
 0xb3,0x19,0x09,0x71,0x53,0x04,0x40,0x80,0x51,0x6b,0xd0,0x83,
 0xbf,0xcc,0xe6,0x61,0x21,0xa3,0x07,0x26,0x46,0x99,0x4c,0x84,
 0x30,0xcc,0x38,0x2b,0x8d,0xc5,0x43,0xe8,
};
const unsigned char tc_ADa[] = {
 0x41,0x44,0x61,
};
const unsigned char tc_Ya[] = {
 0x7f,0x64,0x57,0x72,0xcc,0x20,0x9b,0xf9,0xfd,0x9d,0x76,0xdb,
 0xb1,0x02,0x83,0xbe,0xa7,0x1b,0x12,0x23,0x5e,0x3b,0xb2,0x18,
 0x78,0xd5,0xe5,0x6a,0x70,0x50,0x6e,0x16,0x57,0x43,0xa6,0x32,
 0xde,0x98,0xec,0xa9,0x93,0x2c,0x5d,0x2e,0xfe,0x36,0x50,0x0a,
 0x59,0xb2,0xfd,0xae,0xd0,0xd8,0xa1,0x48,
};
const unsigned char tc_yb[] = {
 0x84,0x8b,0x07,0x79,0xff,0x41,0x5f,0x0a,0xf4,0xea,0x14,0xdf,
 0x9d,0xd1,0xd3,0xc2,0x9a,0xc4,0x1d,0x83,0x6c,0x78,0x08,0x89,
 0x6c,0x4e,0xba,0x19,0xc5,0x1a,0xc4,0x0a,0x43,0x9c,0xaf,0x5e,
 0x61,0xec,0x88,0xc3,0x07,0xc7,0xd6,0x19,0x19,0x52,0x29,0x41,
 0x2e,0xaa,0x73,0xfb,0x2a,0x5e,0xa2,0x0d,
};
const unsigned char tc_ADb[] = {
 0x41,0x44,0x62,
};
const unsigned char tc_Yb[] = {
 0xa4,0x69,0x0a,0x07,0x50,0xc4,0x2b,0x28,0x8d,0xdd,0x0b,0xa0,
 0x8e,0x3f,0x49,0x02,0xdf,0xe7,0x0b,0xae,0x5c,0x9e,0x2c,0x6e,
 0xe9,0x58,0x44,0xde,0xf2,0x69,0x2b,0xe7,0x76,0x46,0xb2,0x0d,
 0x3b,0x42,0x9f,0x8d,0xa0,0x0d,0x21,0x43,0x3e,0xe0,0x89,0x1c,
 0x66,0x76,0x58,0xd8,0xd0,0xc4,0x8e,0x38,
};
const unsigned char tc_K[] = {
 0xdb,0x3f,0xff,0x9d,0xa5,0x95,0x76,0x71,0x5b,0x04,0xd4,0xdf,
 0x8d,0xc8,0xd1,0x8d,0xb2,0x43,0x0e,0x57,0xbb,0xed,0x33,0x7d,
 0xbe,0xee,0x5b,0xb2,0xd6,0xab,0x6c,0xed,0xdc,0x9c,0x75,0xc5,
 0xc0,0xb1,0x7f,0xad,0x7e,0xb7,0x24,0xda,0xa1,0x2f,0x8f,0x19,
```

```
    0x03,0xdd,0x6c,0x2c,0xed,0xe6,0x13,0x5a,
   };
   const unsigned char tc_ISK_IR[] = {
    0x59,0x98,0x92,0xa2,0x07,0x8a,0x8c,0x98,0x81,0x81,0x62,0x5e,
    0x1e,0x5e,0x5f,0x7a,0x61,0x63,0xf7,0xd7,0x2f,0x21,0xb9,0x3e,
    0xbe,0xfb,0xa0,0xf1,0x7f,0xf7,0xea,0x3a,0xa0,0x59,0x4b,0xd5,
    0x69,0xcf,0x74,0x26,0x41,0x57,0xb3,0xc0,0x08,0x7b,0xdc,0xcf,
    0x2f,0x59,0xc7,0x71,0x56,0x62,0x84,0x87,0xf5,0xca,0x16,0x45,
    0xb8,0xe9,0xd0,0x5b,
   };
   const unsigned char tc_ISK_SY[] = {
    0x3a,0xc7,0x3f,0x03,0x03,0x02,0x96,0xaa,0x59,0x1f,0x01,0x32,
    0x6b,0x18,0xaf,0xa4,0x7e,0x11,0x89,0x12,0x9c,0xd0,0x6a,0xe8,
    0xdf,0xb0,0x5e,0x6e,0xb1,0x31,0x0c,0xde,0x94,0x8b,0x59,0xee,
    0xf0,0x75,0x53,0x65,0xc0,0x6a,0x33,0x92,0x66,0xaf,0xe5,0x94,
    0x94,0x8c,0x56,0xa5,0x38,0xd9,0x8a,0x65,0x76,0x71,0x13,0x93,
    0x8a,0x9a,0x78,0xd8,
   };
   const unsigned char tc_ISK_SY[] = {
    0x3a,0xc7,0x3f,0x03,0x03,0x02,0x96,0xaa,0x59,0x1f,0x01,0x32,
    0x6b,0x18,0xaf,0xa4,0x7e,0x11,0x89,0x12,0x9c,0xd0,0x6a,0xe8,
    0xdf,0xb0,0x5e,0x6e,0xb1,0x31,0x0c,0xde,0x94,0x8b,0x59,0xee,
    0xf0,0x75,0x53,0x65,0xc0,0x6a,0x33,0x92,0x66,0xaf,0xe5,0x94,
    0x94,0x8c,0x56,0xa5,0x38,0xd9,0x8a,0x65,0x76,0x71,0x13,0x93,
    0x8a,0x9a,0x78,0xd8,
   };
   const unsigned char tc_sid_out_ir[] = {
    0x00,0xa2,0x33,0x3a,0x79,0x48,0x1a,0xbe,0x71,0xef,0xd6,0x59,
    0x4d,0x7b,0xba,0xac,0x55,0xc8,0x08,0x48,0x2e,0x86,0x9c,0x9b,
    0x65,0xc4,0xb5,0x3d,0x71,0x00,0xd3,0xda,0x8f,0x3c,0xab,0xd5,
    0x9f,0xa0,0xc1,0xf2,0x2d,0x6d,0x2f,0x9a,0xc0,0xc0,0x93,0x96,
    0x22,0x92,0x79,0x8f,0xca,0x2c,0x0b,0x93,0x26,0x89,0x74,0xca,
    0xd7,0x5d,0x57,0x5a,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xa1,0xce,0x90,0x53,0x7a,0x8d,0x53,0xb0,0x6d,0x77,0xe7,0x9f,
    0xe7,0x19,0x46,0x1c,0xc5,0xed,0x83,0x00,0xd2,0x1d,0x18,0x66,
    0xa5,0x9f,0x96,0x38,0x60,0x18,0x33,0xf5,0x7a,0x8b,0x5e,0x88,
    0xdb,0x9a,0x52,0xab,0xfa,0x1b,0x4e,0x8a,0x65,0x1a,0x40,0x0b,
    0xc9,0x20,0x50,0x82,0xaa,0xd8,0x1e,0xb3,0x11,0xc4,0x43,0x73,
    0xb9,0xa1,0x9e,0xff,
   };
```

B.2.9. Testvectors as JSON file encoded as BASE64

```
#eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
#czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI1
#MjIzRTBDREM0NUQ2NTc1NjY4RDY0QzU1MjAwNDEyNCIsICJnIjogIkUyOTNCN0NDRj
#YxQ0E3RUI5MjhBMjYzOTFDRjM4QjY2MEY4NzRBMDAxRkRGMEJGM0E5MUZEMTgyRjJC
#NkQ4M0U2MUE5Mzc3RURFMTI3RUJBN0UwRDRDMDg1OTJFQUZGMzNENEFBNzA1RDZDRT
#U0QkIiLCAieWEiOiAiMjFCNEY0QkQ5RTY0RUQzNTVDM0VCNjc2QTI4RUJFREFGNkQ4
#RjE3QkRDMzY1OTk1QjMxOTA5NzE1MzA0NDA4MDUxNkJEMDgzQkZDQ0U2NjEyMUEzMD
#cyNjQ2OTk0Qzg0MzBDQzM4MkI4REM1NDNFOCIsICJBRGEiOiAiNDE0NDYxIiwgIllh
#IjogIjdGNjQ1NzcyQ0MyMDlCRjlGRDlENzZEQkIxMDI4M0JFQTcxQjEyMjM1RTNCQj
#IxODc4RDVFNTZBNzA1MDZFMTY1NzQzQTYzMkRFOThFQ0E5OTMyQzVEMkVGRTM2NTAw
#QTU5QjJGREFFRDBEOEExNDgiLCAieWIiOiAiODQ4QjA3NzlGRjQxNUYwQUY0RUExNE
#RGOUREMUQzQzI5QUM0MUQ4MzZDNzgwODg5NkM0RUJBMTlDNTFBQzQwQTQzOUNBRjVF
#NjFFQzg4QzMwN0M3RDYxOTE5NTIyOTQxMkVBQTczRkIyQTVFQTIwRCIsICJBRGIiOi
#AiNDE0NDYyIiwgIlliIjogIkE0NjkwQTA3NTBDNDJCMjg4REREMEJBMDhFM0Y0OTAy
#REZFNzBCQUU1QzlFMkM2RUU5NTg0NERFRjI2OTJCRTc3NjQ2QjIwRDNCNDI5RjhEQT
#AwRDIxNDMzRUUwODkxQzY2NzY1OEQ4RDBDNDhFMzgiLCAiSyI6ICJEQjNGRkY5REE1
#OTU3NjcxNUIwNEQ0REY4REM4RDE4REIyNDMwRTU3QkJFRDMzN0RCRUVFNUJCMkQ2QU
#I2Q0VEREM5Qzc1QzVDMEIxN0ZBRDdFQjcyNERBQTEyRjhGMTkwM0RENkMyQ0VERTYx
#MzVBIiwgIklTS19JUiI6ICI1OTk4OTJBMjA3OEE4Qzk4ODE4MTYyNUUxRTVFNUY3QT
#YxNjNGN0Q3MkYyMUI5M0VCRUZCQTBGMTdGRjdFQTNBQTA1OTRCRDU2OUNGNzQyNjQx
#NTdCM0MwMDg3QkRDQ0YyRjU5Qzc3MTU2NjI4NDg3RjVDQTE2NDVCOEU5RDA1QiIsIC
#JJU0tfU1kiOiAiM0FDNzNGMDMwMzAyOTZBQTU5MUYwMTMyNkIxOEFGQTQ3RTExODkx
#MjlDRDA2QUU4REZCMDVFNkVCMTMxMENERTk0OEI1OUVFRjA3NTUzNjVDMDZBMzM5Mj
#Y2QUZFNTk0OTQ4QzU2QTUzOEQ5OEE2NTc2NzExMzkzOEE5QTc4RDgiLCAic2lkX291
#dHB1dF9pciI6ICIwMEEyMzMzQTc5NDgxQUJFNzFFRkQ2NTk0RDdCQkFBQzU1QzgwOD
#Q4MkU4NjlDOUI2NUM0QjUzRDcxMDBEM0RBOEYzQ0FCRDU5RkEwQzFGMjJENkQyRjlB
#QzBDMDkzOTYyMjkyNzk4RkNBMkMwQjkzMjY4OTc0Q0FENzVENTc1QSIsICJzaWRfb3
#V0cHV0X29jIjogIkExQ0U5MDUzN0E4RDUzQjA2RDc3RTc5RkU3MTk0NjFDQzVFRDgz
#MDBEMjFEMTg2NkE1OUY5NjM4NjAxODMzRjU3QThCNUU4OERCOUE1MkFCRkExQjRFOE
#E2NTFBNDAwQkM5MjA1MDgyQUFEODFFQjMxMUM0NDM3M0I5QTE5RUZGIn0=
```

B.2.10. Test vectors for G_X448.scalar_mult_vfy: low order points

Test vectors for which G_X448.scalar_mult_vfy(s_in,ux) must return the neutral element. This includes points that are non-canonicaly encoded, i.e. have coordinate values larger than the field prime.

Weak points for X448 smaller than the field prime (canonical)

```
u0: (length: 56 bytes)
  0000000000000000000000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000000000000000000000
u1: (length: 56 bytes)
  0100000000000000000000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000000000000000000000
u2: (length: 56 bytes)
  fefffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffe
  ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Weak points for X448 larger or equal to the field prime (non-canonical)

u3: (length: 56 bytes)
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffe
ffffffffffffffffffffffffffffffffffffffffffffffffffffffff
u4: (length: 56 bytes)
000000000000000000000000000000000000000000000000000000ff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffff

All of the above points u0 ... u4 MUST trigger the abort case when included in the protocol messages from A or B.

Expected results for X448 resp. G_X448.scalar_mult_vfy

scalar s: (length: 56 bytes)
af8a14218bf2a2062926d2ea9b8fe4e8b6817349b6ed2feb1e5d64d7a4
523f15fceec70fb111e870dc58d191e66a14d3e9d482d04432cadd
G_X448.scalar_mult_vfy(s,u0): (length: 56 bytes)
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
G_X448.scalar_mult_vfy(s,u1): (length: 56 bytes)
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
G_X448.scalar_mult_vfy(s,u2): (length: 56 bytes)
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
G_X448.scalar_mult_vfy(s,u3): (length: 56 bytes)
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
G_X448.scalar_mult_vfy(s,u4): (length: 56 bytes)
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000

Test vectors for scalar_mult with nonzero outputs

```
scalar s: (length: 56 bytes)
  af8a14218bf2a2062926d2ea9b8fe4e8b6817349b6ed2feb1e5d64d7a4
  523f15fceec70fb111e870dc58d191e66a14d3e9d482d04432cadd
point coordinate u_curve on the curve: (length: 56 bytes)
  ab0c68d772ec2eb9de25c49700e46d6325e66d6aa39d7b65eb84a68c55
  69d47bd71b41f3e0d210f44e146dec8926b174acb3f940a0b82cab
G_X448.scalar_mult_vfy(s,u_curve): (length: 56 bytes)
  3b0fa9bc40a6fdc78c9e06ff7a54c143c5d52f365607053bf0656f5142
  0496295f910a101b38edc1acd3bd240fd55dcb7a360553b8a7627e

point coordinate u_twist on the twist: (length: 56 bytes)
  c981cd1e1f72d9c35c7d7cf6be426757c0dc8206a2fcfa564a8e7618c0
  3c0e61f9a2eb1c3e0dd97d6e9b1010f5edd03397a83f5a914cb3ff
G_X448.scalar_mult_vfy(s,u_twist): (length: 56 bytes)
  d0a2bb7e9c5c2c627793d8342f23b759fe7d9e3320a85ca4fd61376331
  50ffd9a9148a9b75c349fac43d64bec49a6e126cc92cbfbf353961
```

B.2.10.1. Testvectors as JSON file encoded as BASE64

```
#eyJJbnZhbGlkIFkxIjogIjAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAiLCAiSW52YWxpZCBZMiI6ICIwMT
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwIiwgIkludmFsaWQgWTMiOiAiRkVGRkZGRkZGRkZGRkZGRkZG
#RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRUZGRkZGRkZGRkZGRk
#ZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRiIsICJJbnZh
#bGlkIFk0IjogIkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRk
#ZGRkZGRkZGRkZGRkZGRkZGRkVGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZG
#RkZGRkZGRkZGRkZGRkZGRkZGRkYiLCAiSW52YWxpZCBZNSI6ICIwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMEZG
#RkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRkZGRk
#ZGRkZGIiwgIlZhbGlkIChvbiBjdXJ2ZSkiOiB7InMiOiAiQUY4QTE0MjE4QkYyQTIw
#NjI5MjZEMkVBOUI4RkU0RThCNjgxNzM0OUI2RUQyRkVCMUU1RDY0RDdBNDUyM0YxNU
#ZDRUVDNzBGQjExMUU4NzBEQzU4RDE5MUU2NkExNEQzRTlENDgyRDA0NDMyQ0FERCIs
#ICJ1X2N1cnZlIjogIkFCMEM2OEQ3NzJFQzJFQjlERTI1QzQ5NzAwRTQ2RDYzMjVFNj
#ZENkFBMzlEN0I2NUVCODRBNjhDNTU2OUQ0N0JENzFCNDFGM0UwRDIxMEY0NEUxNDZE
#RUM4OTI2QjE3NEFDQjNGOTQwQTBCODJDQUIiLCAicmVzX2N1cnZlIjogIjNCMEZBOU
#JDNDBBNkZEQzc4QzlFMDZGRjdBNTRDMTQzQzVENTJGMzY1NjA3MDUzQkYwNjU2RjUx
#NDIwNDk2Mjk1RjkxMEExMDFCMzhFREMxQUNEM0JEMjQwRkQ1NURDQjdBMzYwNTUzQj
#hBNzYyN0UifSwgIlZhbGlkIChvbiB0d2lzdCkiOiB7InMiOiAiQUY4QTE0MjE4QkYy
#QTIwNjI5MjZEMkVBOUI4RkU0RThCNjgxNzM0OUI2RUQyRkVCMUU1RDY0RDdBNDUyM0
#YxNUZDRUVDNzBGQjExMUU4NzBEQzU4RDE5MUU2NkExNEQzRTlENDgyRDA0NDMyQ0FE
#RCIsICJ1X3R3aXN0IjogIkM5ODFDRDFFMUY3MkQ5QzM1QzdEN0NGNkJFNDI2NzU3Qz
#BEQzgyMDZBMkZDRkE1NjRBOEU3NjE4QzAzQzBFNjFGOUEyRUIxQzNFMEREOTdENkU5
#QjEwMTBGNUVERDAzMzk3QTgzRjVBOTE0Q0IzRkYiLCAicmVzX3R3aXN0IjogIkQwQT
#JCQjdFOUM1QzJDNjI3NzkzRDgzNDJGMjNCNzU5RkU3RDlFMzMyMEE4NUNBNEZENjEz
#NzYzMzE1MEZGRDlBOTE0OEE5Qjc1QzM0OUZBQzQzRDY0QkVDNDlBNkUxMjZDQzkyQ0
#JGQkYzNTM5NjEifX0=
```

B.3. Test vector for CPace using group ristretto255 and hash SHA-512

B.3.1. Test vectors for calculate_generator with group ristretto255

```
Inputs
  H   = SHA-512 with input block size 128 bytes.
  PRS = b'Password' ; ZPAD length: 100 ;
  DSI = b'CPaceRistretto255'
  CI = b'oc\x0bB_responder\x0bA_initiator'
  CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
  sid = 7e4b4791d6a8ef019b936c79fb7f2c57
Outputs
  generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
  (length: 172 bytes)
    1143506163655269737472657474 6f32353085061737377 6f726464
    0000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000001a6f630b425f726573706f6e
    6465720b415f696e69746961746f72107e4b4791d6a8ef019b936c79
    fb7f2c57
  hash result: (length: 64 bytes)
    c63a5750e2439c17ccd8213be14fde2f87e1bc637001a97f5929c77b
    30ea0e08afbc75ace5d3d73b2842a79d01488c5fd7ea30d475ee6095
    45af1bfd1ff77c8e
  encoded generator g: (length: 32 bytes)
    a6fc82c3b8968fbb2e06fee81ca858586dea50d248f0c7ca6a18b090
    2a30b36b
```

B.3.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS01MTIiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
#E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiAxMDAsICJEU0kiOiAiNDM1MDYx
#NjM2NTUyNjk3Mzc0NzI2NTc0NzQ2RjMyMzUzNSIsICJDSSI6ICI2RjYzMEI0MjVGNz
#I2NTczNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6
#ICI3RTRCNDc5MUQ2QThFRjAxOUI5MzZDNzlGQjdGMkM1NyIsICJnZW5lcmF0b3Jfc3
#RyaW5nKEcuRFNJLFBSUyxDSSxzaWQsSC5zX2luX2J5dGVzKSI6ICIxMTQzNTA2MTYz
#NjU1MjY5NzM3NDcyNjU3NDc0NkYzMjM1MzUwODUwNjE3MzczNzc2RjcyNjQ2NDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMU
#E2RjYzMEI0MjVGNzI2NTczNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2
#RjcyMTA3RTRCNDc5MUQ2QThFRjAxOUI5MzZDNzlGQjdGMkM1NyIsICJoYXNoIHJlc3
#VsdCI6ICJDNjNBNTc1MEUyNDM5QzE3Q0NEODIxM0JFMTRGREUyRjg3RTFCQzYzNzAw
#MUE5N0Y1OTI5Qzc3QjMwRUEwRTA4QUZCQzc1QUNFNUQzRDczQjI4NDJBNzlEMDE0OD
#hDNUZEN0VBMzBENDc1RUU2MDk1NDVBRjFCRkQxRkY3N0M4RSIsICJlbmNvZGVkIGdl
#bmVyYXRvciBnIjogIkE2RkM4MkMzQjg5NjhGQkIyRTA2RkVFODFDQTg1ODU4NkRFQT
#UwRDI0OEYwQzdDQTZBMThCMDkwMkEzMEIzNkIifQ==
```

B.3.2. Test vector for message from A

```
     Inputs
       ADa = b'ADa'
       ya (little endian): (length: 32 bytes)
         da3d23700a9e5699258aef94dc060dfda5ebb61f02a5ea77fad53f4f
         f0976d08
     Outputs
       Ya: (length: 32 bytes)
         d40fb265a7abeaee7939d91a585fe59f7053f982c296ec413c624c66
         9308f87a
```

B.3.3. Test vector for message from B

```
     Inputs
       ADb = b'ADb'
       yb (little endian): (length: 32 bytes)
         d2316b454718c35362d83d69df6320f38578ed5984651435e2949762
         d900b80d
     Outputs
       Yb: (length: 32 bytes)
         08bcf6e9777a9c313a3db6daa510f2d398403319c2341bd506a92e67
         2eb7e307
```

B.3.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 32 bytes)
         e22b1ef7788f661478f3cddd4c600774fc0f41e6b711569190ff88fa
         0e607e09
       scalar_mult_vfy(yb,Ya): (length: 32 bytes)
         e22b1ef7788f661478f3cddd4c600774fc0f41e6b711569190ff88fa
         0e607e09
```

B.3.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 74 bytes)
  20d40fb265a7abeaee7939d91a585fe59f7053f982c296ec413c624c
  669308f87a034144612008bcf6e9777a9c313a3db6daa510f2d39840
  3319c2341bd506a92e672eb7e30703414462
DSI = G.DSI_ISK, b'CPaceRistretto255_ISK':
(length: 21 bytes)
  435061636552697374726574746f3235355f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 146 bytes)
  154350616365526973747265747 46f3235355f49534b107e4b4791d6
  a8ef019b936c79fb7f2c5720e22b1ef7788f661478f3cddd4c600774
  fc0f41e6b711569190ff88fa0e607e0920d40fb265a7abeaee7939d9
  1a585fe59f7053f982c296ec413c624c669308f87a034144612008bc
  f6e9777a9c313a3db6daa510f2d398403319c2341bd506a92e672eb7
  e30703414462
ISK result: (length: 64 bytes)
  4c5469a16b2364c4b944ebc1a79e51d1674ad47db26e8718154f59fa
  ebfaa52d8346f30aa58377117eb20d527f2cbc5c76381f7fd372e89d
  f8239f87f2e02ed1
```

B.3.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 76 bytes)
  6f6320d40fb265a7abeaee7939d91a585fe59f7053f982c296ec413c
  624c669308f87a034144612008bcf6e9777a9c313a3db6daa510f2d3
  98403319c2341bd506a92e672eb7e30703414462
DSI = G.DSI_ISK, b'CPaceRistretto255_ISK':
(length: 21 bytes)
  435061636552697374726574746f3235355f49534b
lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
(length: 148 bytes)
  154350616365526973747265747 46f3235355f49534b107e4b4791d6
  a8ef019b936c79fb7f2c5720e22b1ef7788f661478f3cddd4c600774
  fc0f41e6b711569190ff88fa0e607e096f6320d40fb265a7abeaee79
  39d91a585fe59f7053f982c296ec413c624c669308f87a0341446120
  08bcf6e9777a9c313a3db6daa510f2d398403319c2341bd506a92e67
  2eb7e30703414462
ISK result: (length: 64 bytes)
  980dcc5a1c52ceea031e75f38ed266586616488c5c5780285fcbcf79
  087c7bcdbd993502eee606b718ba31e840a000a7b7befe15ea427c5c
  fe88344fa1237f35
```

B.3.7. Test vector for optional output of session id

```
      H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        2a76d3bbc499dfdc4dcacc9ff042f4e1a54e3843258e100ccd7c60f0
        a541f9d3ebf025e68a460dde218bd39f0711bc6fa11409c9d7b69d8c
        cf6b32fc51ddb699
      H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        ca4b50700c46203ccd10bc0e9f31095e508189cb59857537be561048
        d34b9ed9a9697af11c998f484c3d783b0b531434caa6835d4c32344f
        cd17160c9c348fc7
```

B.3.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
    0x7e,0x4b,0x47,0x91,0xd6,0xa8,0xef,0x01,0x9b,0x93,0x6c,0x79,
    0xfb,0x7f,0x2c,0x57,
   };
   const unsigned char tc_g[] = {
    0xa6,0xfc,0x82,0xc3,0xb8,0x96,0x8f,0xbb,0x2e,0x06,0xfe,0xe8,
    0x1c,0xa8,0x58,0x58,0x6d,0xea,0x50,0xd2,0x48,0xf0,0xc7,0xca,
    0x6a,0x18,0xb0,0x90,0x2a,0x30,0xb3,0x6b,
   };
   const unsigned char tc_ya[] = {
    0xda,0x3d,0x23,0x70,0x0a,0x9e,0x56,0x99,0x25,0x8a,0xef,0x94,
    0xdc,0x06,0x0d,0xfd,0xa5,0xeb,0xb6,0x1f,0x02,0xa5,0xea,0x77,
    0xfa,0xd5,0x3f,0x4f,0xf0,0x97,0x6d,0x08,
   };
   const unsigned char tc_ADa[] = {
    0x41,0x44,0x61,
   };
   const unsigned char tc_Ya[] = {
    0xd4,0x0f,0xb2,0x65,0xa7,0xab,0xea,0xee,0x79,0x39,0xd9,0x1a,
    0x58,0x5f,0xe5,0x9f,0x70,0x53,0xf9,0x82,0xc2,0x96,0xec,0x41,
    0x3c,0x62,0x4c,0x66,0x93,0x08,0xf8,0x7a,
   };
   const unsigned char tc_yb[] = {
    0xd2,0x31,0x6b,0x45,0x47,0x18,0xc3,0x53,0x62,0xd8,0x3d,0x69,
    0xdf,0x63,0x20,0xf3,0x85,0x78,0xed,0x59,0x84,0x65,0x14,0x35,
    0xe2,0x94,0x97,0x62,0xd9,0x00,0xb8,0x0d,
   };
```

```
   const unsigned char tc_ADb[] = {
    0x41,0x44,0x62,
   };
   const unsigned char tc_Yb[] = {
    0x08,0xbc,0xf6,0xe9,0x77,0x7a,0x9c,0x31,0x3a,0x3d,0xb6,0xda,
    0xa5,0x10,0xf2,0xd3,0x98,0x40,0x33,0x19,0xc2,0x34,0x1b,0xd5,
    0x06,0xa9,0x2e,0x67,0x2e,0xb7,0xe3,0x07,
   };
   const unsigned char tc_K[] = {
    0xe2,0x2b,0x1e,0xf7,0x78,0x8f,0x66,0x14,0x78,0xf3,0xcd,0xdd,
    0x4c,0x60,0x07,0x74,0xfc,0x0f,0x41,0xe6,0xb7,0x11,0x56,0x91,
    0x90,0xff,0x88,0xfa,0x0e,0x60,0x7e,0x09,
   };
   const unsigned char tc_ISK_IR[] = {
    0x4c,0x54,0x69,0xa1,0x6b,0x23,0x64,0xc4,0xb9,0x44,0xeb,0xc1,
    0xa7,0x9e,0x51,0xd1,0x67,0x4a,0xd4,0x7d,0xb2,0x6e,0x87,0x18,
    0x15,0x4f,0x59,0xfa,0xeb,0xfa,0xa5,0x2d,0x83,0x46,0xf3,0x0a,
    0xa5,0x83,0x77,0x11,0x7e,0xb2,0x0d,0x52,0x7f,0x2c,0xbc,0x5c,
    0x76,0x38,0x1f,0x7f,0xd3,0x72,0xe8,0x9d,0xf8,0x23,0x9f,0x87,
    0xf2,0xe0,0x2e,0xd1,
   };
   const unsigned char tc_ISK_SY[] = {
    0x98,0x0d,0xcc,0x5a,0x1c,0x52,0xce,0xea,0x03,0x1e,0x75,0xf3,
    0x8e,0xd2,0x66,0x58,0x66,0x16,0x48,0x8c,0x5c,0x57,0x80,0x28,
    0x5f,0xcb,0xcf,0x79,0x08,0x7c,0x7b,0xcd,0xbd,0x99,0x35,0x02,
    0xee,0xe6,0x06,0xb7,0x18,0xba,0x31,0xe8,0x40,0xa0,0x00,0xa7,
    0xb7,0xbe,0xfe,0x15,0xea,0x42,0x7c,0x5c,0xfe,0x88,0x34,0x4f,
    0xa1,0x23,0x7f,0x35,
   };
   const unsigned char tc_ISK_SY[] = {
    0x98,0x0d,0xcc,0x5a,0x1c,0x52,0xce,0xea,0x03,0x1e,0x75,0xf3,
    0x8e,0xd2,0x66,0x58,0x66,0x16,0x48,0x8c,0x5c,0x57,0x80,0x28,
    0x5f,0xcb,0xcf,0x79,0x08,0x7c,0x7b,0xcd,0xbd,0x99,0x35,0x02,
    0xee,0xe6,0x06,0xb7,0x18,0xba,0x31,0xe8,0x40,0xa0,0x00,0xa7,
    0xb7,0xbe,0xfe,0x15,0xea,0x42,0x7c,0x5c,0xfe,0x88,0x34,0x4f,
    0xa1,0x23,0x7f,0x35,
   };
   const unsigned char tc_sid_out_ir[] = {
    0x2a,0x76,0xd3,0xbb,0xc4,0x99,0xdf,0xdc,0x4d,0xca,0xcc,0x9f,
    0xf0,0x42,0xf4,0xe1,0xa5,0x4e,0x38,0x43,0x25,0x8e,0x10,0x0c,
    0xcd,0x7c,0x60,0xf0,0xa5,0x41,0xf9,0xd3,0xeb,0xf0,0x25,0xe6,
    0x8a,0x46,0x0d,0xde,0x21,0x8b,0xd3,0x9f,0x07,0x11,0xbc,0x6f,
    0xa1,0x14,0x09,0xc9,0xd7,0xb6,0x9d,0x8c,0xcf,0x6b,0x32,0xfc,
    0x51,0xdd,0xb6,0x99,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xca,0x4b,0x50,0x70,0x0c,0x46,0x20,0x3c,0xcd,0x10,0xbc,0x0e,
    0x9f,0x31,0x09,0x5e,0x50,0x81,0x89,0xcb,0x59,0x85,0x75,0x37,
```

```
    0xbe,0x56,0x10,0x48,0xd3,0x4b,0x9e,0xd9,0xa9,0x69,0x7a,0xf1,
    0x1c,0x99,0x8f,0x48,0x4c,0x3d,0x78,0x3b,0x0b,0x53,0x14,0x34,
    0xca,0xa6,0x83,0x5d,0x4c,0x32,0x34,0x4f,0xcd,0x17,0x16,0x0c,
    0x9c,0x34,0x8f,0xc7,
   };
```

B.3.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
    #czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI3
    #RTRCNDc5MUQ2QThFRjAxOUI5MzZDNzlGQjdGMkM1NyIsICJnIjogIkE2RkM4MkMzQj
    #g5NjhGQkIyRTA2RkVFODFDQTg1ODU4NkRFQTUwRDI0OEYwQzdDQTZBMThCMDkwMkEz
    #MEIzNkIiLCAieWEiOiAiREEzRDIzNzAwQTlFNTY5OTI1OEFFRjk0REMwNjBERkRBNU
    #VCQjYxRjAyQTVFQTc3RkFENTNGNEZGMDk3NkQwOCIsICJBRGEiOiAiNDE0NDYxIiwg
    #IllhIjogIkQ0MEZCMjY1QTdBQkVBRUU3OTM5RDkxQTU4NUZFNTlGNzA1M0Y5ODJDMj
    #k2RUM0MTNDNjI0QzY2OTMwOEY4N0EiLCAieWIiOiAiRDIzMTZCNDU0NzE4QzM1MzYy
    #RDgzRDY5REY2MzIwRjM4NTc4RUQ1OTg0NjUxNDM1RTI5NDk3NjJEOTAwQjgwRCIsIC
    #JBRGIiOiAiNDE0NDYyIiwgIlliIjogIjA4QkNGNkU5Nzc3QTlDMzEzQTNEQjZEQUE1
    #MTBGMkQzOTg0MDMzMTlDMjM0MUJENTA2QTkyRTY3MkVCN0UzMDciLCAiSyI6ICJFMj
    #JCMUVGNzc4OEY2NjE0NzhGM0NEREQ0QzYwMDc3NEZDMEY0MUU2QjcxMTU2OTE5MEZG
    #ODhGQTBFNjA3RTA5IiwgIklTS19JUiI6ICI0QzU0NjlBMTZCMjM2NEM0Qjk0NEVCQz
    #FBNzlFNTFEMTY3NEFENDdEQjI2RTg3MTgxNTRGNTlGQUVCRkFBNTJEODM0NkYzMEFB
    #NTgzNzcxMTdFQjIwRDUyN0YyQ0JDNUM3NjM4MUY3RkQzNzJFODlERjgyMzlGODdGMk
    #UwMkVEMSIsICJJU0tfU1kiOiAiOTgwRENDNUExQzUyQ0VFQTAzMUU3NUYzOEVEMjY2
    #NTg2NjE2NDg4QzVDNTc4MDI4NUZDQkNGNzkwODdDN0JDREJEOTkzNTAyRUVFNjA2Qj
    #cxOEJBMzFFODQwQTAwMEE3QjdCRUZFMTVFQTQyN0M1Q0ZFODgzNDRGQTEyMzdGMzUi
    #LCAic2lkX291dHB1dF9pciI6ICIyQTc2RDNCQkM0OTlERkRDNERDQUNDOUZGMDQyRj
    #RFMUE1NEUzODQzMjU4RTEwMENDRDdDNjBGMEE1NDFGOUQzRUJGMDI1RTY4QTQ2MERE
    #RTIxOEJEMzlGMDcxMUJDNkZBMTE0MDlDOUQ3QjY5RDhDQ0Y2QjMyRkM1MUREQjY5OS
    #IsICJzaWRfb3V0cHV0X29jIjogIkNBNEI1MDcwMEM0NjIwM0NDRDEwQkMwRTlGMzEw
    #OTVFNTA4MTg5Q0I1OTg1NzUzN0JFNTYxMDQ4RDM0QjlFRDlBOTY5N0FGMTFDOTk4Rj
    #Q4NEMzRDc4M0IwQjUzMTQzNENBQTY4MzVENEMzMjM0NEZDRDE3MTYwQzlDMzQ4RkM3
    #In0=
```

B.3.10. Test case for scalar_mult with valid inputs

```
   s: (length: 32 bytes)
     7cd0e075fa7955ba52c02759a6c90dbbfc10e6d40aea8d283e407d88
     cf538a05
   X: (length: 32 bytes)
     2c3c6b8c4f3800e7aef6864025b4ed79bd599117e427c41bd47d93d6
     54b4a51c
   G.scalar_mult(s,decode(X)): (length: 32 bytes)
     7c13645fe790a468f62c39beb7388e541d8405d1ade69d1778c5fe3e
     7f6b600e
   G.scalar_mult_vfy(s,X): (length: 32 bytes)
     7c13645fe790a468f62c39beb7388e541d8405d1ade69d1778c5fe3e
     7f6b600e
```

B.3.11. Invalid inputs for scalar_mult_vfy

For these test cases scalar_mult_vfy(y,.) MUST return the representation of the neutral element G.I. When points Y_i1 or Y_i2 are included in message of A or B the protocol MUST abort.

s: (length: 32 bytes)
7cd0e075fa7955ba52c02759a6c90dbbfc10e6d40aea8d283e407d88
cf538a05
Y_i1: (length: 32 bytes)
2b3c6b8c4f3800e7aef6864025b4ed79bd599117e427c41bd47d93d6
54b4a51c
Y_i2 == G.I: (length: 32 bytes)
0000000000000000000000000000000000000000000000000000000000000000
00000000
G.scalar_mult_vfy(s,Y_i1) = G.scalar_mult_vfy(s,Y_i2) = G.I

B.3.11.1. Testvectors as JSON file encoded as BASE64

```
#eyJWYWxpZCI6IHsicyI6ICI3Q0QwRTA3NUZBNzk1NUJBNTJDMDI3NTlBNkM5MERCQk
#ZDMTBFNkQ0MEFFQThEMjgzRTQwN0Q4OENGNTM4QTA1IiwgIlgiOiAiMkMzQzZCOEM0
#RjM4MDBFN0FFRjY4NjQwMjVCNEVENzlCRDU5OTExN0U0MjdDNDFCRDQ3RDkzRDY1NE
#I0QTUxQyIsICJHLnNjYWxhcl9tdWx0KHMsZGVjb2RlKFgpKSI6ICI3QzEzNjQ1RkU3
#OTBBNDY4RjYyQzM5QkVCNzM4OEU1NDFEODQwNUQxQURFNjlEMTc3OEM1RkUzRTdGNk
#I2MDBFIiwgIkcuc2NhbGFyX211bHRfdmZ5KHMsWCkiOiAiN0MxMzY0NUZFNzkwQTQ2
#OEY2MkMzOUJFQjczODhFNTQxRDg0MDVEMUFERTY5RDE3NzhDNUZFM0U3RjZCNjAwRS
#J9LCAiSW52YWxpZCBZMSI6ICIyQjNDNkI4QzRGMzgwMEU3QUVGNjg2NDAyNUI0RUQ3
#OUJENTk5MTE3RTQyN0M0MUJENDdEOTNENjU0QjRBNTFDIiwgIkludmFsaWQgWTIiOi
#AiMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMCJ9
```

B.4. Test vector for CPace using group decaf448 and hash SHAKE-256

B.4.1. Test vectors for calculate_generator with group decaf448

```
     Inputs
       H   = SHAKE-256 with input block size 136 bytes.
       PRS = b'Password' ; ZPAD length: 112 ;
       DSI = b'CPaceDecaf448'
       CI = b'oc\x0bB_responder\x0bA_initiator'
       CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
       sid = 5223e0cdc45d6575668d64c552004124
     Outputs
       generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
       (length: 180 bytes)
         0d435061636544656361663434380850617373776f72647000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000001a6f630b
         425f726573706f6e6465720b415f696e69746961746f72105223e0cd
         c45d6575668d64c552004124
       hash result: (length: 112 bytes)
         7148f4d60587aaafa64d2fd6bcfe45ee71e8b971d1d5ff3bbf8c1451
         797c62a1af22ab25638749f97f9b15fedcf4aeee87282cf667594ab0
         92b6023c8f8d3a61c38b0af791c9271137df01b57d63b79734bbce69
         91e3e2e10fdc805abc9e6e6f3daeff6fd34093d26de240b326764252
       encoded generator g: (length: 56 bytes)
         9a700ecc378eb98e57387df456d5b4b4f1dceebbb1371527eeb7e1bf
         bab64ecc9c9303396145ba04f5b5aea5baedfa61f31f00fbc5fd5606
```

B.4.1.1. Testvectors as JSON file encoded as BASE64

```
    #eyJIIjogIlNIQUtFLTI1NiIsICJILnNfaW5fYnl0ZXMiOiAxMzYsICJQUlMiOiAiNT
    #A2MTczNzM3NzZGNzI2NCIsICJaUEFEIGxlbmd0aCI6IDExMiwgIkRTSSI6ICI0MzUw
    #NjE2MzY1NDQ2NTYzNjE2NjM0MzQzOCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NTczNz
    #A2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI1MjIz
    #RTBDREM0NUQ2NTc1NjY4RDY0QzU1MjAwNDEyNCIsICJnZW5lcmF0b3Jfc3RyaW5nKE
    #cuRFNJLFBSUyxDSSxzaWQsSC5zX2luX2J5dGVzKSI6ICIwRDQzNTA2MTYzNjU0NDY1
    #NjM2MTY2MzQzNDM4MDg1MDYxNzM3Mzc3NkY3MjY0NzAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
    #MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
    #MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDFBNkY2MzBCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjBCNDE1RjY5NkU2
    #OTc0Njk2MTc0NkY3MjEwNTIyM0UwQ0RDNDVENjU3NTY2OEQ2NEM1NTIwMDQxMjQiLC
    #AiaGFzaCByZXN1bHQiOiAiNzE0OEY0RDYwNTg3QUFBRkE2NEQyRkQ2QkNGRTQ1RUU3
    #MUU4Qjk3MUQxRDVGRjNCQkY4QzE0NTE3OTdDNjJBMUFGMjJBQjI1NjM4NzQ5Rjk3Rj
    #lCMTVGRURDRjRBRUVFODcyODJDRjY2NzU5NEFCMDkyQjYwMjNDOEY4RDNBNjFDMzhC
    #MEFGNzkxQzkyNzExMzdERjAxQjU3RDYzQjc5NzM0QkJDRTY5OTFFM0UyRTEwRkRDOD
    #A1QUJDOUU2RTZGM0RBRUZGNkZEMzQwOTNEMjZERTI0MEIzMjY3NjQyNTIiLCAiZW5j
    #b2RlZCBnZW5lcmF0b3IgZyI6ICI5QTcwMEVDQzM3OEVCOThFNTczODdERjQ1NkQ1Qj
    #RCNEYxRENFRUJCQjEzNzE1MjdFRUI3RTFCRkJBQjY0RUNDOUM5MzAzMzk2MTQ1QkEw
    #NEY1QjVBRUE1QkFFREZBNjFGMzFGMDBGQkM1RkQ1NjA2In0=
```

B.4.2. Test vector for message from A

```
Inputs
  ADa = b’ADa’
  ya (little endian): (length: 56 bytes)
    33d561f13cfc0dca279c30e8cde895175dc25483892819eba132d58c
    13c0462a8eb0d73fda941950594bef5191d8394691f86edffcad6c1e
Outputs
  Ya: (length: 56 bytes)
    627f8bb2ae945e2a518967df9b00aff19253d3086398f2ec18be846c
    c0d1f286c2ce3caf1da639859ccd2a6a01a9372a17e66bb7006e571b
```

B.4.3. Test vector for message from B

```
Inputs
  ADb = b’ADb’
  yb (little endian): (length: 56 bytes)
    2523c969f68fa2b2aea294c2539ef36eb1e0558abd14712a7828f16a
    85ed2c7e77e2bdd418994405fb1b57b6bbaadd66849892aac9d81402
Outputs
  Yb: (length: 56 bytes)
    8e9811e4402fac098743ca7b2b509b91b38c8cf1360cc6cab3011871
    7019782b7f58a591c63d9c9247b774e6b0e0b826ff4f8399f94772db
```

B.4.4. Test vector for secret points K

```
scalar_mult_vfy(ya,Yb): (length: 56 bytes)
  94f4ae494f4e8b07ade3354726eee49c5518b363cda544f5b4541b97
  32830be37ea0e63fc83f54be280dea0747a043c76d473e01689af77f
scalar_mult_vfy(yb,Ya): (length: 56 bytes)
  94f4ae494f4e8b07ade3354726eee49c5518b363cda544f5b4541b97
  32830be37ea0e63fc83f54be280dea0747a043c76d473e01689af77f
```

B.4.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 122 bytes)
  38627f8bb2ae945e2a518967df9b00aff19253d3086398f2ec18be84
  6cc0d1f286c2ce3caf1da639859ccd2a6a01a9372a17e66bb7006e57
  1b03414461388e9811e4402fac098743ca7b2b509b91b38c8cf1360c
  c6cab30118717019782b7f58a591c63d9c9247b774e6b0e0b826ff4f
  8399f94772db03414462
DSI = G.DSI_ISK, b'CPaceDecaf448_ISK': (length: 17 bytes)
  435061636544656361663434385f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 214 bytes)
  1143506163654465636166343438 5f49534b105223e0cdc45d657566
  8d64c5520041243894f4ae494f4e8b07ade3354726eee49c5518b363
  cda544f5b4541b9732830be37ea0e63fc83f54be280dea0747a043c7
  6d473e01689af77f38627f8bb2ae945e2a518967df9b00aff19253d3
  086398f2ec18be846cc0d1f286c2ce3caf1da639859ccd2a6a01a937
  2a17e66bb7006e571b03414461388e9811e4402fac098743ca7b2b50
  9b91b38c8cf1360cc6cab30118717019782b7f58a591c63d9c9247b7
  74e6b0e0b826ff4f8399f94772db03414462
ISK result: (length: 64 bytes)
  9c2726a6cda1179349cbc38f31765eab646a2a5f176f3019fab4a0aa
  bd9d17c2ba895998cff698d801761a003512c1cf67d144b21e1cb6d6
  b82da71d0da76cad
```

B.4.6. Test vector for ISK calculation parallel execution

```
       transcript_oc(Ya,ADa,Yb,ADb): (length: 124 bytes)
         6f63388e9811e4402fac098743ca7b2b509b91b38c8cf1360cc6cab3
         0118717019782b7f58a591c63d9c9247b774e6b0e0b826ff4f8399f9
         4772db0341446238627f8bb2ae945e2a518967df9b00aff19253d308
         6398f2ec18be846cc0d1f286c2ce3caf1da639859ccd2a6a01a9372a
         17e66bb7006e571b03414461
       DSI = G.DSI_ISK, b'CPaceDecaf448_ISK': (length: 17 bytes)
         435061636544656361663434385f49534b
       lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
       (length: 216 bytes)
         1143506163654465636166343438 5f49534b105223e0cdc45d657566
         8d64c5520041243894f4ae494f4e8b07ade3354726eee49c5518b363
         cda544f5b4541b9732830be37ea0e63fc83f54be280dea0747a043c7
         6d473e01689af77f6f63388e9811e4402fac098743ca7b2b509b91b3
         8c8cf1360cc6cab30118717019782b7f58a591c63d9c9247b774e6b0
         e0b826ff4f8399f94772db0341446238627f8bb2ae945e2a518967df
         9b00aff19253d3086398f2ec18be846cc0d1f286c2ce3caf1da63985
         9ccd2a6a01a9372a17e66bb7006e571b03414461
       ISK result: (length: 64 bytes)
         6d2178ed3048703025b9007ec84c4d969e8d8135df455e608c16aa15
         2e1219c86cea563254428a9d969903ae3649d1050da1e6e0c1c060e1
         ebf7316a7e993389
```

B.4.7. Test vector for optional output of session id

```
       H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         65bffde17b7acf07cfd437bdb973a8f7340bf911d393a61498c0a50e
         f0d68bca103fbdb0f5b799505562e59811df1bc5d9b4f5f0f7c57c22
         cd7ed6db4d153e3a
       H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         7ce27043d1b1d0d0e02e16979637e2a00547ed6e15ea988f7d3c9b3c
         2159b26ab3834bff7ff86240323e25216ba2ee6ea6e1582502017f8e
         6d65f8c4a5e65543
```

B.4.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
    0x52,0x23,0xe0,0xcd,0xc4,0x5d,0x65,0x75,0x66,0x8d,0x64,0xc5,
```

```
 0x52,0x00,0x41,0x24,
};
const unsigned char tc_g[] = {
 0x9a,0x70,0x0e,0xcc,0x37,0x8e,0xb9,0x8e,0x57,0x38,0x7d,0xf4,
 0x56,0xd5,0xb4,0xb4,0xf1,0xdc,0xee,0xbb,0xb1,0x37,0x15,0x27,
 0xee,0xb7,0xe1,0xbf,0xba,0xb6,0x4e,0xcc,0x9c,0x93,0x03,0x39,
 0x61,0x45,0xba,0x04,0xf5,0xb5,0xae,0xa5,0xba,0xed,0xfa,0x61,
 0xf3,0x1f,0x00,0xfb,0xc5,0xfd,0x56,0x06,
};
const unsigned char tc_ya[] = {
 0x33,0xd5,0x61,0xf1,0x3c,0xfc,0x0d,0xca,0x27,0x9c,0x30,0xe8,
 0xcd,0xe8,0x95,0x17,0x5d,0xc2,0x54,0x83,0x89,0x28,0x19,0xeb,
 0xa1,0x32,0xd5,0x8c,0x13,0xc0,0x46,0x2a,0x8e,0xb0,0xd7,0x3f,
 0xda,0x94,0x19,0x50,0x59,0x4b,0xef,0x51,0x91,0xd8,0x39,0x46,
 0x91,0xf8,0x6e,0xdf,0xfc,0xad,0x6c,0x1e,
};
const unsigned char tc_ADa[] = {
 0x41,0x44,0x61,
};
const unsigned char tc_Ya[] = {
 0x62,0x7f,0x8b,0xb2,0xae,0x94,0x5e,0x2a,0x51,0x89,0x67,0xdf,
 0x9b,0x00,0xaf,0xf1,0x92,0x53,0xd3,0x08,0x63,0x98,0xf2,0xec,
 0x18,0xbe,0x84,0x6c,0xc0,0xd1,0xf2,0x86,0xc2,0xce,0x3c,0xaf,
 0x1d,0xa6,0x39,0x85,0x9c,0xcd,0x2a,0x6a,0x01,0xa9,0x37,0x2a,
 0x17,0xe6,0x6b,0xb7,0x00,0x6e,0x57,0x1b,
};
const unsigned char tc_yb[] = {
 0x25,0x23,0xc9,0x69,0xf6,0x8f,0xa2,0xb2,0xae,0xa2,0x94,0xc2,
 0x53,0x9e,0xf3,0x6e,0xb1,0xe0,0x55,0x8a,0xbd,0x14,0x71,0x2a,
 0x78,0x28,0xf1,0x6a,0x85,0xed,0x2c,0x7e,0x77,0xe2,0xbd,0xd4,
 0x18,0x99,0x44,0x05,0xfb,0x1b,0x57,0xb6,0xbb,0xaa,0xdd,0x66,
 0x84,0x98,0x92,0xaa,0xc9,0xd8,0x14,0x02,
};
const unsigned char tc_ADb[] = {
 0x41,0x44,0x62,
};
const unsigned char tc_Yb[] = {
 0x8e,0x98,0x11,0xe4,0x40,0x2f,0xac,0x09,0x87,0x43,0xca,0x7b,
 0x2b,0x50,0x9b,0x91,0xb3,0x8c,0x8c,0xf1,0x36,0x0c,0xc6,0xca,
 0xb3,0x01,0x18,0x71,0x70,0x19,0x78,0x2b,0x7f,0x58,0xa5,0x91,
 0xc6,0x3d,0x9c,0x92,0x47,0xb7,0x74,0xe6,0xb0,0xe0,0xb8,0x26,
 0xff,0x4f,0x83,0x99,0xf9,0x47,0x72,0xdb,
};
const unsigned char tc_K[] = {
 0x94,0xf4,0xae,0x49,0x4f,0x4e,0x8b,0x07,0xad,0xe3,0x35,0x47,
 0x26,0xee,0xe4,0x9c,0x55,0x18,0xb3,0x63,0xcd,0xa5,0x44,0xf5,
 0xb4,0x54,0x1b,0x97,0x32,0x83,0x0b,0xe3,0x7e,0xa0,0xe6,0x3f,
 0xc8,0x3f,0x54,0xbe,0x28,0x0d,0xea,0x07,0x47,0xa0,0x43,0xc7,
```

```
    0x6d,0x47,0x3e,0x01,0x68,0x9a,0xf7,0x7f,
   };
   const unsigned char tc_ISK_IR[] = {
    0x9c,0x27,0x26,0xa6,0xcd,0xa1,0x17,0x93,0x49,0xcb,0xc3,0x8f,
    0x31,0x76,0x5e,0xab,0x64,0x6a,0x2a,0x5f,0x17,0x6f,0x30,0x19,
    0xfa,0xb4,0xa0,0xaa,0xbd,0x9d,0x17,0xc2,0xba,0x89,0x59,0x98,
    0xcf,0xf6,0x98,0xd8,0x01,0x76,0x1a,0x00,0x35,0x12,0xc1,0xcf,
    0x67,0xd1,0x44,0xb2,0x1e,0x1c,0xb6,0xd6,0xb8,0x2d,0xa7,0x1d,
    0x0d,0xa7,0x6c,0xad,
   };
   const unsigned char tc_ISK_SY[] = {
    0x6d,0x21,0x78,0xed,0x30,0x48,0x70,0x30,0x25,0xb9,0x00,0x7e,
    0xc8,0x4c,0x4d,0x96,0x9e,0x8d,0x81,0x35,0xdf,0x45,0x5e,0x60,
    0x8c,0x16,0xaa,0x15,0x2e,0x12,0x19,0xc8,0x6c,0xea,0x56,0x32,
    0x54,0x42,0x8a,0x9d,0x96,0x99,0x03,0xae,0x36,0x49,0xd1,0x05,
    0x0d,0xa1,0xe6,0xe0,0xc1,0xc0,0x60,0xe1,0xeb,0xf7,0x31,0x6a,
    0x7e,0x99,0x33,0x89,
   };
   const unsigned char tc_ISK_SY[] = {
    0x6d,0x21,0x78,0xed,0x30,0x48,0x70,0x30,0x25,0xb9,0x00,0x7e,
    0xc8,0x4c,0x4d,0x96,0x9e,0x8d,0x81,0x35,0xdf,0x45,0x5e,0x60,
    0x8c,0x16,0xaa,0x15,0x2e,0x12,0x19,0xc8,0x6c,0xea,0x56,0x32,
    0x54,0x42,0x8a,0x9d,0x96,0x99,0x03,0xae,0x36,0x49,0xd1,0x05,
    0x0d,0xa1,0xe6,0xe0,0xc1,0xc0,0x60,0xe1,0xeb,0xf7,0x31,0x6a,
    0x7e,0x99,0x33,0x89,
   };
   const unsigned char tc_sid_out_ir[] = {
    0x65,0xbf,0xfd,0xe1,0x7b,0x7a,0xcf,0x07,0xcf,0xd4,0x37,0xbd,
    0xb9,0x73,0xa8,0xf7,0x34,0x0b,0xf9,0x11,0xd3,0x93,0xa6,0x14,
    0x98,0xc0,0xa5,0x0e,0xf0,0xd6,0x8b,0xca,0x10,0x3f,0xbd,0xb0,
    0xf5,0xb7,0x99,0x50,0x55,0x62,0xe5,0x98,0x11,0xdf,0x1b,0xc5,
    0xd9,0xb4,0xf5,0xf0,0xf7,0xc5,0x7c,0x22,0xcd,0x7e,0xd6,0xdb,
    0x4d,0x15,0x3e,0x3a,
   };
   const unsigned char tc_sid_out_oc[] = {
    0x7c,0xe2,0x70,0x43,0xd1,0xb1,0xd0,0xd0,0xe0,0x2e,0x16,0x97,
    0x96,0x37,0xe2,0xa0,0x05,0x47,0xed,0x6e,0x15,0xea,0x98,0x8f,
    0x7d,0x3c,0x9b,0x3c,0x21,0x59,0xb2,0x6a,0xb3,0x83,0x4b,0xff,
    0x7f,0xf8,0x62,0x40,0x32,0x3e,0x25,0x21,0x6b,0xa2,0xee,0x6e,
    0xa6,0xe1,0x58,0x25,0x02,0x01,0x7f,0x8e,0x6d,0x65,0xf8,0xc4,
    0xa5,0xe6,0x55,0x43,
   };
```

B.4.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
    #czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI1
    #MjIzRTBDREM0NUQ2NTc1NjY4RDY0QzU1MjAwNDEyNCIsICJnIjogIjlBNzAwRUNDMz
    #c4RUI5OEU1NzM4N0RGNDU2RDVCNEI0RjFEQ0VFQkJCMTM3MTUyN0VFQjdFMUJGQkFC
    #NjRFQ0M5QzkzMDMzOTYxNDVCQTA0RjVCNUFFQTVCQUVERkE2MUYzMUYwMEZCQzVGRD
    #U2MDYiLCAieWEiOiAiMzNENTYxRjEzQ0ZDMERDQTI3OUMzMEU4Q0RFODk1MTc1REMy
    #NTQ4Mzg5MjgxOUVCQTEzMkQ1OEMxM0MwNDYyQThFQjBENzNGREE5NDE5NTA1OTRCRU
    #Y1MTkxRDgzOTQ2OTFGODZFREZGQ0FENkMxRSIsICJBRGEiOiAiNDE0NDYxIiwgIllh
    #IjogIjYyN0Y4QkIyQUU5NDVFMkE1MTg5NjdERjlCMDBBRkYxOTI1M0QzMDg2Mzk4Rj
    #JFQzE4QkU4NDZDQzBEMUYyODZDMkNFM0NBRjFEQTYzOTg1OUNDRDJBNkEwMUE5Mzcy
    #QTE3RTY2QkI3MDA2RTU3MUIiLCAieWIiOiAiMjUyM0M5NjlGNjhGQTJCMkFFQTI5NE
    #MyNTM5RUYzNkVCMUUwNTU4QUJEMTQ3MTJBNzgyOEYxNkE4NUVEMkM3RTc3RTJCREQ0
    #MTg5OTQ0MDVGQjFCNTdCNkJCQUFERDY2ODQ5ODkyQUFDOUQ4MTQwMiIsICJBRGIiOi
    #AiNDE0NDYyIiwgIlliIjogIjhFOTgxMUU0NDAyRkFDMDk4NzQzQ0E3QjJCNTA5Qjkx
    #QjM4QzhDRjEzNjBDQzZDQUIzMDExODcxNzAxOTc4MkI3RjU4QTU5MUM2M0Q5QzkyND
    #dCNzc0RTZCMEUwQjgyNkZGNEY4Mzk5Rjk0NzcyREIiLCAiSyI6ICI5NEY0QUU0OTRG
    #NEU4QjA3QURFMzM1NDcyNkVFRTQ5QzU1MThCMzYzQ0RBNTQ0RjVCNDU0MUI5NzMyOD
    #MwQkUzN0VBMEU2M0ZDODNGNTRCRTI4MERFQTA3NDdBMDQzQzc2RDQ3M0UwMTY4OUFG
    #NzdGIiwgIklTS19JUiI6ICI5QzI3MjZBNkNEQTExNzkzNDlDQkMzOEYzMTc2NUVBQj
    #Y0NkEyQTVGMTc2RjMwMTlGQUI0QTBBQUJEOUQxN0MyQkE4OTU5OThDRkY2OThEODAx
    #NzYxQTAwMzUxMkMxQ0Y2N0QxNDRCMjFFMUNCNkQ2QjgyREE3MUQwREE3NkNBRCIsIC
    #JJU0tfU1kiOiAiNkQyMTc4RUQzMDQ4NzAzMDI1QjkwMDdFQzg0QzREOTY5RThEODEz
    #NURGNDU1RTYwOEMxNkFBMTUyRTEyMTlDODZDRUE1NjMyNTQ0MjhBOUQ5Njk5MDNBRT
    #M2NDlEMTA1MERBMUU2RTBDMUMwNjBFMUVCRjczMTZBN0U5OTMzODkiLCAic2lkX291
    #dHB1dF9pciI6ICI2NUJGRkRFMTdCN0FDRjA3Q0ZENDM3QkRCOTczQThGNzM0MEJGOT
    #ExRDM5M0E2MTQ5OEMwQTUwRUYwRDY4QkNBMTAzRkJEQjBGNUI3OTk1MDU1NjJFNTk4
    #MTFERjFCQzVEOUI0RjVGMEY3QzU3QzIyQ0Q3RUQ2REI0RDE1M0UzQSIsICJzaWRfb3
    #V0cHV0X29jIjogIjdDRTI3MDQzRDFCMUQwRDBFMDJFMTY5Nzk2MzdFMkEwMDU0N0VE
    #NkUxNUVBOTg4RjdEM0M5QjNDMjE1OUIyNkFCMzgzNEJGRjdGRjg2MjQwMzIzRTI1Mj
    #E2QkEyRUU2RUE2RTE1ODI1MDIwMTdGOEU2RDY1RjhDNEE1RTY1NTQzIn0=
```

B.4.10. Test case for scalar_mult with valid inputs

```
    s: (length: 56 bytes)
      dd1bc7015daabb7672129cc35a3ba815486b139deff9bdeca7a4fc61
      34323d34658761e90ff079972a7ca8aa5606498f4f4f0ebc0933a819
    X: (length: 56 bytes)
      601431d5e51f43d422a92d3fb2373bde28217aab42524c341aa404ea
      ba5aa5541f7042dbb3253ce4c90f772b038a413dcb3a0f6bf3ae9e21
    G.scalar_mult(s,decode(X)): (length: 56 bytes)
      388b35c60eb41b66085a2118316218681d78979d667702de105fdc1f
      21ffe884a577d795f45691781390a229a3bd7b527e831380f2f585a4
    G.scalar_mult_vfy(s,X): (length: 56 bytes)
      388b35c60eb41b66085a2118316218681d78979d667702de105fdc1f
      21ffe884a577d795f45691781390a229a3bd7b527e831380f2f585a4
```

B.4.11. Invalid inputs for scalar_mult_vfy

For these test cases scalar_mult_vfy(y,.) MUST return the representation of the neutral element G.I. When points Y_i1 or Y_i2 are included in message of A or B the protocol MUST abort.

```
s: (length: 56 bytes)
  dd1bc7015daabb7672129cc35a3ba815486b139deff9bdeca7a4fc61
  34323d34658761e90ff079972a7ca8aa5606498f4f4f0ebc0933a819
Y_i1: (length: 56 bytes)
  5f1431d5e51f43d422a92d3fb2373bde28217aab42524c341aa404ea
  ba5aa5541f7042dbb3253ce4c90f772b038a413dcb3a0f6bf3ae9e21
Y_i2 == G.I: (length: 56 bytes)
  00000000000000000000000000000000000000000000000000000000
  00000000000000000000000000000000000000000000000000000000
G.scalar_mult_vfy(s,Y_i1) = G.scalar_mult_vfy(s,Y_i2) = G.I
```

B.4.11.1. Testvectors as JSON file encoded as BASE64

```
#eyJWYWxpZCI6IHsicyI6ICJERDFCQzcwMTVEQUFCQjc2NzIxMjlDQzM1QTNCQTgxNT
#Q4NkIxMzlERUZGOUJERUNBN0E0RkM2MTM0MzIzRDM0NjU4NzYxRTkwRkYwNzk5NzJB
#N0NBOEFBNTYwNjQ5OEY0RjRGMEVCQzA5MzNBODE5IiwgIlgiOiAiNjAxNDMxRDVFNT
#FGNDNENDIyQTkyRDNGQjIzNzNCREUyODIxN0FBQjQyNTI0QzM0MUFBNDA0RUFCQTVB
#QTU1NDFGNzA0MkRCQjMyNTNDRTRDOTBGNzcyQjAzOEE0MTNEQ0IzQTBGNkJGM0FFOU
#UyMSIsICJHLnNjYWxhcl9tdWx0KHMsZGVjb2RlKFgpKSI6ICIzODhCMzVDNjBFQjQx
#QjY2MDg1QTIxMTgzMTYyMTg2ODFENzg5NzlENjY3NzAyREUxMDVGREMxRjIxRkZFOD
#g0QTU3N0Q3OTVGNDU2OTE3ODEzOTBBMjI5QTNCRDdCNTI3RTgzMTM4MEYyRjU4NUE0
#IiwgIkcuc2NhbGFyX211bHRfdmZ5KHMsWCkiOiAiMzg4QjM1QzYwRUI0MUI2NjA4NU
#EyMTE4MzE2MjE4NjgxRDc4OTc5RDY2NzcwMkRFMTA1RkRDMUYyMUZGRTg4NEE1NzdE
#Nzk1RjQ1NjkxNzgxMzkwQTIyOUEzQkQ3QjUyN0U4MzEzODBGMkY1ODVBNCJ9LCAiSW
#52YWxpZCBZMSI6ICI1RjE0MzFENUU1MUY0M0Q0MjJBOTJEM0ZCMjM3M0JERTI4MjE3
#QUFCNDI1MjRDMzQxQUE0MDRFQUJBNUFBNTU0MUY3MDQyREJCMzI1M0NFNEM5MEY3Nz
#JCMDM4QTQxM0RDQjNBMEY2QkYzQUU5RTIxIiwgIkludmFsaWQgWTIiOiAiMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMCJ9
```

B.5. Test vector for CPace using group NIST P-256 and hash SHA-256

B.5.1. Test vectors for calculate_generator with group NIST P-256

```
     Inputs
       H   = SHA-256 with input block size 64 bytes.
       PRS = b'Password' ; ZPAD length: 23 ;
       DSI = b'CPaceP256_XMD:SHA-256_SSWU_NU_'
       DST = b'CPaceP256_XMD:SHA-256_SSWU_NU__DST'
       CI = b'oc\x0bB_responder\x0bA_initiator'
       CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
       sid = 34b36454cab2e7842c389f7d88ecb7df
     Outputs
       generator_string(PRS,G.DSI,CI,sid,H.s_in_bytes):
       (length: 108 bytes)
         1e435061636550323536 5f584d443a5348412d3235365f535357555f
         4e555f0850617373776f726417000000000000000000000000000000
         0000000000000001a6f630b425f726573706f6e6465720b415f696e
         69746961746f721034b36454cab2e7842c389f7d88ecb7df
       generator g: (length: 65 bytes)
         04eee577320b1c241a79419fcde5718c2b63f81ef8717d56a57d2fb2
         6b65a8beb63573b52605efb32ff4cf31aaef9a92df84e4e8408cc6c7
         cf27a535aad2b38a56
```

B.5.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS0yNTYiLCAiSC5zX2luX2J5dGVzIjogNjQsICJQUlMiOiAiNTA2MT
#czNzM3NzZGNzI2NCIsICJaUEFEIGxlbmd0aCI6IDIzLCAiRFNJIjogIjQzNTA2MTYz
#NjU1MDMyMzUzNjVGNTg0RDQ0M0E1MzQ4NDEyRDMyMzUzNjVGNTM1MzU3NTU1RjRFNT
#U1RiIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NTczNzA2RjZFNjQ2NTcyMEI0MTVGNjk2
#RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICIzNEIzNjQ1NENBQjJFNzg0MkMzODlGN0
#Q4OEVDQjdERiIsICJnZW5lcmF0b3Jfc3RyaW5nKEcuRFNJLFBSUyxDSSxzaWQsSC5z
#X2luX2J5dGVzKSI6ICIxRTQzNTA2MTYzNjU1MDMyMzUzNjVGNTg0RDQ0M0E1MzQ4ND
#EyRDMyMzUzNjVGNTM1MzU3NTU1RjRFNTU1RjA4NTA2MTczNzM3NzZGNzI2NDE3MDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDFBNkY2Mz
#BCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjBCNDE1RjY5NkU2OTc0Njk2MTc0NkY3MjEw
#MzRCMzY0NTRDQUIyRTc4NDJDMzg5RjdEODhFQ0I3REYiLCAiZ2VuZXJhdG9yIGciOi
#AiMDRFRUU1NzczMjBCMUMyNDFBNzk0MTlGQ0RFNTcxOEMyQjYzRjgxRUY4NzE3RDU2
#QTU3RDJGQjI2QjY1QThCRUI2MzU3M0I1MjYwNUVGQjMyRkY0Q0YzMUFBRUY5QTkyRE
#Y4NEU0RTg0MDhDQzZDN0NGMjdBNTM1QUFEMkIzOEE1NiJ9
```

B.5.2. Test vector for message from A

```
     Inputs
       ADa = b'ADa'
       ya (big endian): (length: 32 bytes)
         37574cfbf1b95ff6a8e2d7be462d4d01e6dde2618f34f4de9df869b2
         4f532c5d
     Outputs
       Ya: (length: 65 bytes)
         041f12ad5fc65010a24fc04c86197109a36df0e9ce85a7479e1e1364
         692fdace17ea5a634e19c207a5d52ead6c6817a163cf2f2fe3406c5d
         fdfc2ecdf8e42c5e16
       Alternative correct value for Ya: (-ya)*g:
       (length: 65 bytes)
         041f12ad5fc65010a24fc04c86197109a36df0e9ce85a7479e1e1364
         692fdace1715a59cb0e63df85b2ad1529397e85e9c30d0d01dbf93a2
         0203d132071bd3a1e9
```

B.5.3. Test vector for message from B

```
     Inputs
       ADb = b'ADb'
       yb (big endian): (length: 32 bytes)
         e5672fc9eb4e721f41d80181ec4c9fd9886668acc48024d33c82bb10
         2aecba52
     Outputs
       Yb: (length: 65 bytes)
         046a51180b6ebabaf5ed0af8cd786886d93342bcae4c158ce1617a0a
         cc8ec354486f9ed2e9210913206d1e3f5e463d2d320c4f1f5ce8b677
         a7e38a26f752bf8f84
       Alternative correct value for Yb: (-yb)*g:
       (length: 65 bytes)
         046a51180b6ebabaf5ed0af8cd786886d93342bcae4c158ce1617a0a
         cc8ec3544890612d15def6ece092e1c0a1b9c2d2cdf3b0e0a4174988
         581c75d908ad40707b
```

B.5.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 32 bytes)
         3e0e2f8976fb8d0deee30aef4b5cd3631eed249af32f53d0dd009b5d
         7b8f6b6c
       scalar_mult_vfy(yb,Ya): (length: 32 bytes)
         3e0e2f8976fb8d0deee30aef4b5cd3631eed249af32f53d0dd009b5d
         7b8f6b6c
```

B.5.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 140 bytes)
  41041f12ad5fc65010a24fc04c86197109a36df0e9ce85a7479e1e13
  64692fdace17ea5a634e19c207a5d52ead6c6817a163cf2f2fe3406c
  5dfdfc2ecdf8e42c5e160341446141046a51180b6ebabaf5ed0af8cd
  786886d93342bcae4c158ce1617a0acc8ec354486f9ed2e921091320
  6d1e3f5e463d2d320c4f1f5ce8b677a7e38a26f752bf8f8403414462
DSI = G.DSI_ISK, b'CPaceP256_XMD:SHA-256_SSWU_NU__ISK':
(length: 34 bytes)
  4350616365503235365f584d443a5348412d3235365f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 225 bytes)
  2243506163655032353 65f584d443a5348412d3235365f535357555f
  4e555f5f49534b1034b36454cab2e7842c389f7d88ecb7df203e0e2f
  8976fb8d0deee30aef4b5cd3631eed249af32f53d0dd009b5d7b8f6b
  6c41041f12ad5fc65010a24fc04c86197109a36df0e9ce85a7479e1e
  1364692fdace17ea5a634e19c207a5d52ead6c6817a163cf2f2fe340
  6c5dfdfc2ecdf8e42c5e160341446141046a51180b6ebabaf5ed0af8
  cd786886d93342bcae4c158ce1617a0acc8ec354486f9ed2e9210913
  206d1e3f5e463d2d320c4f1f5ce8b677a7e38a26f752bf8f84034144
  62
ISK result: (length: 32 bytes)
  9565ed286b6e3cf1f943fd31746f9a22935537025a1328d4980005ba
  984f0c39
```

B.5.6. Test vector for ISK calculation parallel execution

```
       transcript_oc(Ya,ADa,Yb,ADb): (length: 142 bytes)
         6f6341046a51180b6ebabaf5ed0af8cd786886d93342bcae4c158ce1
         617a0acc8ec354486f9ed2e9210913206d1e3f5e463d2d320c4f1f5c
         e8b677a7e38a26f752bf8f840341446241041f12ad5fc65010a24fc0
         4c86197109a36df0e9ce85a7479e1e1364692fdace17ea5a634e19c2
         07a5d52ead6c6817a163cf2f2fe3406c5dfdfc2ecdf8e42c5e160341
         4461
       DSI = G.DSI_ISK, b'CPaceP256_XMD:SHA-256_SSWU_NU__ISK':
       (length: 34 bytes)
         435061636550323536 5f584d443a5348412d3235365f535357555f4e
         555f5f49534b
       lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
       (length: 227 bytes)
         224350616365503235365f584d443a5348412d3235365f535357555f
         4e555f5f49534b1034b36454cab2e7842c389f7d88ecb7df203e0e2f
         8976fb8d0deee30aef4b5cd3631eed249af32f53d0dd009b5d7b8f6b
         6c6f6341046a51180b6ebabaf5ed0af8cd786886d93342bcae4c158c
         e1617a0acc8ec354486f9ed2e9210913206d1e3f5e463d2d320c4f1f
         5ce8b677a7e38a26f752bf8f840341446241041f12ad5fc65010a24f
         c04c86197109a36df0e9ce85a7479e1e1364692fdace17ea5a634e19
         c207a5d52ead6c6817a163cf2f2fe3406c5dfdfc2ecdf8e42c5e1603
         414461
       ISK result: (length: 32 bytes)
         62a445a4daa3546dd031c66ead2e4e015abbcc83bde31c90f841149f
         d441c58a
```

B.5.7. Test vector for optional output of session id

```
       H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
       (length: 32 bytes)
         a7386d1c2eb06e8056d7fecbcf691e08d189d96236020ef31b414069
         8a4b99f9
       H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
       (length: 32 bytes)
         d337dbc0dd797b4e6f2f14ea4925c58e5d5523871e8cb43a3c1b0f2b
         1a1ffde3
```

B.5.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
```

```
    0x34,0xb3,0x64,0x54,0xca,0xb2,0xe7,0x84,0x2c,0x38,0x9f,0x7d,
    0x88,0xec,0xb7,0xdf,
   };
   const unsigned char tc_g[] = {
    0x04,0xee,0xe5,0x77,0x32,0x0b,0x1c,0x24,0x1a,0x79,0x41,0x9f,
    0xcd,0xe5,0x71,0x8c,0x2b,0x63,0xf8,0x1e,0xf8,0x71,0x7d,0x56,
    0xa5,0x7d,0x2f,0xb2,0x6b,0x65,0xa8,0xbe,0xb6,0x35,0x73,0xb5,
    0x26,0x05,0xef,0xb3,0x2f,0xf4,0xcf,0x31,0xaa,0xef,0x9a,0x92,
    0xdf,0x84,0xe4,0xe8,0x40,0x8c,0xc6,0xc7,0xcf,0x27,0xa5,0x35,
    0xaa,0xd2,0xb3,0x8a,0x56,
   };
   const unsigned char tc_ya[] = {
    0x37,0x57,0x4c,0xfb,0xf1,0xb9,0x5f,0xf6,0xa8,0xe2,0xd7,0xbe,
    0x46,0x2d,0x4d,0x01,0xe6,0xdd,0xe2,0x61,0x8f,0x34,0xf4,0xde,
    0x9d,0xf8,0x69,0xb2,0x4f,0x53,0x2c,0x5d,
   };
   const unsigned char tc_ADa[] = {
    0x41,0x44,0x61,
   };
   const unsigned char tc_Ya[] = {
    0x04,0x1f,0x12,0xad,0x5f,0xc6,0x50,0x10,0xa2,0x4f,0xc0,0x4c,
    0x86,0x19,0x71,0x09,0xa3,0x6d,0xf0,0xe9,0xce,0x85,0xa7,0x47,
    0x9e,0x1e,0x13,0x64,0x69,0x2f,0xda,0xce,0x17,0xea,0x5a,0x63,
    0x4e,0x19,0xc2,0x07,0xa5,0xd5,0x2e,0xad,0x6c,0x68,0x17,0xa1,
    0x63,0xcf,0x2f,0x2f,0xe3,0x40,0x6c,0x5d,0xfd,0xfc,0x2e,0xcd,
    0xf8,0xe4,0x2c,0x5e,0x16,
   };
   const unsigned char tc_yb[] = {
    0xe5,0x67,0x2f,0xc9,0xeb,0x4e,0x72,0x1f,0x41,0xd8,0x01,0x81,
    0xec,0x4c,0x9f,0xd9,0x88,0x66,0x68,0xac,0xc4,0x80,0x24,0xd3,
    0x3c,0x82,0xbb,0x10,0x2a,0xec,0xba,0x52,
   };
   const unsigned char tc_ADb[] = {
    0x41,0x44,0x62,
   };
   const unsigned char tc_Yb[] = {
    0x04,0x6a,0x51,0x18,0x0b,0x6e,0xba,0xba,0xf5,0xed,0x0a,0xf8,
    0xcd,0x78,0x68,0x86,0xd9,0x33,0x42,0xbc,0xae,0x4c,0x15,0x8c,
    0xe1,0x61,0x7a,0x0a,0xcc,0x8e,0xc3,0x54,0x48,0x6f,0x9e,0xd2,
    0xe9,0x21,0x09,0x13,0x20,0x6d,0x1e,0x3f,0x5e,0x46,0x3d,0x2d,
    0x32,0x0c,0x4f,0x1f,0x5c,0xe8,0xb6,0x77,0xa7,0xe3,0x8a,0x26,
    0xf7,0x52,0xbf,0x8f,0x84,
   };
   const unsigned char tc_K[] = {
    0x3e,0x0e,0x2f,0x89,0x76,0xfb,0x8d,0x0d,0xee,0xe3,0x0a,0xef,
    0x4b,0x5c,0xd3,0x63,0x1e,0xed,0x24,0x9a,0xf3,0x2f,0x53,0xd0,
    0xdd,0x00,0x9b,0x5d,0x7b,0x8f,0x6b,0x6c,
   };
```

```
   const unsigned char tc_ISK_IR[] = {
    0x95,0x65,0xed,0x28,0x6b,0x6e,0x3c,0xf1,0xf9,0x43,0xfd,0x31,
    0x74,0x6f,0x9a,0x22,0x93,0x55,0x37,0x02,0x5a,0x13,0x28,0xd4,
    0x98,0x00,0x05,0xba,0x98,0x4f,0x0c,0x39,
   };
   const unsigned char tc_ISK_SY[] = {
    0x62,0xa4,0x45,0xa4,0xda,0xa3,0x54,0x6d,0xd0,0x31,0xc6,0x6e,
    0xad,0x2e,0x4e,0x01,0x5a,0xbb,0xcc,0x83,0xbd,0xe3,0x1c,0x90,
    0xf8,0x41,0x14,0x9f,0xd4,0x41,0xc5,0x8a,
   };
   const unsigned char tc_ISK_SY[] = {
    0x62,0xa4,0x45,0xa4,0xda,0xa3,0x54,0x6d,0xd0,0x31,0xc6,0x6e,
    0xad,0x2e,0x4e,0x01,0x5a,0xbb,0xcc,0x83,0xbd,0xe3,0x1c,0x90,
    0xf8,0x41,0x14,0x9f,0xd4,0x41,0xc5,0x8a,
   };
   const unsigned char tc_sid_out_ir[] = {
    0xa7,0x38,0x6d,0x1c,0x2e,0xb0,0x6e,0x80,0x56,0xd7,0xfe,0xcb,
    0xcf,0x69,0x1e,0x08,0xd1,0x89,0xd9,0x62,0x36,0x02,0x0e,0xf3,
    0x1b,0x41,0x40,0x69,0x8a,0x4b,0x99,0xf9,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xd3,0x37,0xdb,0xc0,0xdd,0x79,0x7b,0x4e,0x6f,0x2f,0x14,0xea,
    0x49,0x25,0xc5,0x8e,0x5d,0x55,0x23,0x87,0x1e,0x8c,0xb4,0x3a,
    0x3c,0x1b,0x0f,0x2b,0x1a,0x1f,0xfd,0xe3,
   };
```

B.5.9. Testvectors as JSON file encoded as BASE64

```
#eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
#czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICIz
#NEIzNjQ1NENBQjJFNzg0MkMzODlGN0Q4OEVDQjdERiIsICJnIjogIjA0RUVFNTc3Mz
#IwQjFDMjQxQTc5NDE5RkNERTU3MThDMkI2M0Y4MUVGODcxN0Q1NkE1N0QyRkIyNkI2
#NUE4QkVCNjM1NzNCNTI2MDVFRkIzMkZGNENGMzFBQUVGOUE5MkRGODRFNEU4NDA4Q0
#M2QzdDRjI3QTUzNUFBRDJCMzhBNTYiLCAieWEiOiAiMzc1NzRDRkJGMUI5NUZGNkE4
#RTJEN0JFNDYyRDREMDFFNkRERTI2MThGMzRGNERFOURGODY5QjI0RjUzMkM1RCIsIC
#JBRGEiOiAiNDE0NDYxIiwgIllhIjogIjA0MUYxMkFENUZDNjUwMTBBMjRGQzA0Qzg2
#MTk3MTA5QTM2REYwRTlDRTg1QTc0NzlFMUUxMzY0NjkyRkRBQ0UxN0VBNUE2MzRFMT
#lDMjA3QTVENTJFQUQ2QzY4MTdBMTYzQ0YyRjJGRTM0MDZDNURGREZDMkVDREY4RTQy
#QzVFMTYiLCAieWIiOiAiRTU2NzJGQzlFQjRFNzIxRjQxRDgwMTgxRUM0QzlGRDk4OD
#Y2NjhBQ0M0ODAyNEQzM0M4MkJCMTAyQUVDQkE1MiIsICJBRGIiOiAiNDE0NDYyIiwg
#IlliIjogIjA0NkE1MTE4MEI2RUJBQkFGNUVEMEFGOENENzg2ODg2RDkzMzQyQkNBRT
#RDMTU4Q0UxNjE3QTBBQ0M4RUMzNTQ0ODZGOUVEMkU5MjEwOTEzMjA2RDFFM0Y1RTQ2
#M0QyRDMyMEM0RjFGNUNFOEI2NzdBN0UzOEEyNkY3NTJCRjhGODQiLCAiSyI6ICIzRT
#BFMkY4OTc2RkI4RDBERUVFMzBBRUY0QjVDRDM2MzFFRUQyNDlBRjMyRjUzRDBERDAw
#OUI1RDdCOEY2QjZDIiwgIklTS19JUiI6ICI5NTY1RUQyODZCNkUzQ0YxRjk0M0ZEMz
#E3NDZGOUEyMjkzNTUzNzAyNUExMzI4RDQ5ODAwMDVCQTk4NEYwQzM5IiwgIklTS19T
#WSI6ICI2MkE0NDVBNERBQTM1NDZERDAzMUM2NkVBRDJFNEUwMTVBQkJDQzgzQkRFMz
#FDOTBGODQxMTQ5RkQ0NDFDNThBIiwgInNpZF9vdXRwdXRfaXIiOiAiQTczODZEMUMy
#RUIwNkU4MDU2RDdGRUNCQ0Y2OTFFMDhEMTg5RDk2MjM2MDIwRUYzMUI0MTQwNjk4QT
#RCOTlGOSIsICJzaWRfb3V0cHV0X29jIjogIkQzMzdEQkMwREQ3OTdCNEU2RjJGMTRF
#QTQ5MjVDNThFNUQ1NTIzODcxRThDQjQzQTNDMUIwRjJCMUExRkZERTMifQ==
```

B.5.10. Test case for scalar_mult_vfy with correct inputs

```
s: (length: 32 bytes)
  f012501c091ff9b99a123fffe571d8bc01e8077ee581362e1bd21399
  0835643b
X: (length: 65 bytes)
  0424648eb986c2be0af636455cef0550671d6bcd8aa26e0d72ffa1b1
  fd12ba4e0f78da2b6d2184f31af39e566aef127014b6936c9a37346d
  10a4ab2514faef5831
G.scalar_mult(s,X) (full coordinates): (length: 65 bytes)
  04f5a191f078c87c36633b78c701751159d56c59f3fe9105b5720673
  470f303ab925b6a7fd1cdd8f649a21cf36b68d9e9c4a11919a951892
  519786104b27033757
G.scalar_mult_vfy(s,X) (only X-coordinate):
(length: 32 bytes)
  f5a191f078c87c36633b78c701751159d56c59f3fe9105b572067347
  0f303ab9
```

B.5.11. Invalid inputs for scalar_mult_vfy

For these test cases scalar_mult_vfy(y,.) MUST return the representation of the neutral element G.I. When including Y_i1 or Y_i2 in messages of A or B the protocol MUST abort.

```
      s: (length: 32 bytes)
        f012501c091ff9b99a123fffe571d8bc01e8077ee581362e1bd21399
        0835643b
      Y_i1: (length: 65 bytes)
        0424648eb986c2be0af636455cef0550671d6bcd8aa26e0d72ffa1b1
        fd12ba4e0f78da2b6d2184f31af39e566aef127014b6936c9a37346d
        10a4ab2514faef5857
      Y_i2: (length: 1 bytes)
        00
      G.scalar_mult_vfy(s,Y_i1) = G.scalar_mult_vfy(s,Y_i2) = G.I
```

B.5.11.1. Testvectors as JSON file encoded as BASE64

```
    #eyJWYWxpZCI6IHsicyI6ICJGMDEyNTAxQzA5MUZGOUI5OUExMjNGRkZFNTcxRDhCQz
    #AxRTgwNzdFRTU4MTM2MkUxQkQyMTM5OTA4MzU2NDNCIiwgIlgiOiAiMDQyNDY0OEVC
    #OTg2QzJCRTBBRjYzNjQ1NUNFRjA1NTA2NzFENkJDRDhBQTI2RTBENzJGRkExQjFGRD
    #EyQkE0RTBGNzhEQTJCNkQyMTg0RjMxQUYzOUU1NjZBRUYxMjcwMTRCNjkzNkM5QTM3
    #MzQ2RDEwQTRBQjI1MTRGQUVGNTgzMSIsICJHLnNjYWxhcl9tdWx0KHMsWCkgKGZ1bG
    #wgY29vcmRpbmF0ZXMpIjogIjA0RjVBMTkxRjA3OEM4N0MzNjYzM0I3OEM3MDE3NTEx
    #NTlENTZDNTlGM0ZFOTEwNUI1NzIwNjczNDcwRjMwM0FCOTI1QjZBN0ZEMUNERDhGNj
    #Q5QTIxQ0YzNkI2OEQ5RTlDNEExMTkxOUE5NTE4OTI1MTk3ODYxMDRCMjcwMzM3NTci
    #LCAiRy5zY2FsYXJfbXVsdF92ZnkocyxYKSAob25seSBYLWNvb3JkaW5hdGUpIjogIk
    #Y1QTE5MUYwNzhDODdDMzY2MzNCNzhDNzAxNzUxMTU5RDU2QzU5RjNGRTkxMDVCNTcy
    #MDY3MzQ3MEYzMDNBQjkifSwgIkludmFsaWQgWTEiOiAiMDQyNDY0OEVCOTg2QzJCRT
    #BBRjYzNjQ1NUNFRjA1NTA2NzFENkJDRDhBQTI2RTBENzJGRkExQjFGRDEyQkE0RTBG
    #NzhEQTJCNkQyMTg0RjMxQUYzOUU1NjZBRUYxMjcwMTRCNjkzNkM5QTM3MzQ2RDEwQT
    #RBQjI1MTRGQUVGNTg1NyIsICJJbnZhbGlkIFkyIjogIjAwIn0=
```

B.6. Test vector for CPace using group NIST P-384 and hash SHA-384

B.6.1. Test vectors for calculate_generator with group NIST P-384

```
Inputs
  H   = SHA-384 with input block size 128 bytes.
  PRS = b'Password' ; ZPAD length: 87 ;
  DSI = b'CPaceP384_XMD:SHA-384_SSWU_NU_'
  DST = b'CPaceP384_XMD:SHA-384_SSWU_NU__DST'
  CI = b'oc\x0bB_responder\x0bA_initiator'
  CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
  sid = 5b3773aa90e8f23c61563a4b645b276c
Outputs
  generator_string(PRS,G.DSI,CI,sid,H.s_in_bytes):
  (length: 172 bytes)
    1e435061636550333834 5f584d443a5348412d3338345f535357555f
    4e555f0850617373776f726457000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000001a6f630b425f726573706f6e
    6465720b415f696e69746961746f72105b3773aa90e8f23c61563a4b
    645b276c
  generator g: (length: 97 bytes)
    04ffe1bdc3293fdbe31b2959916e52c018e923eac99836bd9a1cbeec
    794a8d4d78baa32cdafc9685bc1067a780f4ad9c8a6c6e164aa42906
    d1e27f782581adc8e0109219626a2b8fbdc34602e4084554bdd6c0c6
    98dd657ac8e31b2bcce1c7b0d8
```

B.6.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS0zODQiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
#E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiA4NywgIkRTSSI6ICI0MzUwNjE2
#MzY1NTAzMzM4MzQ1RjU4NEQ0NDNBNTM0ODQxMkQzMzM4MzQ1RjUzNTM1NzU1NUY0RT
#U1NUYiLCAiQ0kiOiAiNkY2MzBCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjBCNDE1RjY5
#NkU2OTc0Njk2MTc0NkY3MiIsICJzaWQiOiAiNUIzNzczQUE5MEU4RjIzQzYxNTYzQT
#RCNjQ1QjI3NkMiLCAiZ2VuZXJhdG9yX3N0cmluZyhHLkRTSSxQUlMsQ0ksc2lkLEgu
#c19pbl9ieXRlcykiOiAiMUU0MzUwNjE2MzY1NTAzMzM4MzQ1RjU4NEQ0NDNBNTM0OD
#QxMkQzMzM4MzQ1RjUzNTM1NzU1NUY0RTU1NUYwODUwNjE3MzczNzc2RjcyNjQ1NzAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDFBNkY2MzBCNDI1RjcyNjU3MzcwNkY2RTY0
#NjU3MjBCNDE1RjY5NkU2OTc0Njk2MTc0NkY3MjEwNUIzNzczQUE5MEU4RjIzQzYxNT
#YzQTRCNjQ1QjI3NkMiLCAiZ2VuZXJhdG9yIGciOiAiMDRGRkUxQkRDMzI5M0ZEQkUz
#MUIyOTU5OTE2RTUyQzAxOEU5MjNFQUM5OTgzNkJEOUExQ0JFRUM3OTRBOEQ0RDc4Qk
#FBMzJDREFGQzk2ODVCQzEwNjdBNzgwRjRBRDlDOEE2QzZFMTY0QUE0MjkwNkQxRTI3
#Rjc4MjU4MUFEQzhFMDEwOTIxOTYyNkEyQjhGQkRDMzQ2MDJFNDA4NDU1NEJERDZDME
#M2OThERDY1N0FDOEUzMUIyQkNDRTFDN0IwRDgifQ==
```

B.6.2. Test vector for message from A

```
     Inputs
       ADa = b'ADa'
       ya (big endian): (length: 48 bytes)
         ef433dd5ad142c860e7cb6400dd315d388d5ec5420c550e9d6f0907f
         375d988bc4d704837e43561c497e7dd93edcdb9d
     Outputs
       Ya: (length: 97 bytes)
         0434a04da121995d81d7c5ded02cf2e70954dca49059485ea38310b7
         3b96fa1ec78619c5a1e524472778331fbe009fa1564a3203e42993a3
         6285fa54e6c24184fd227458c87781be16fff46dbc970cc1b1770050
         a94d6826d52f211b234792e66d
       Alternative correct value for Ya: (-ya)*g:
       (length: 97 bytes)
         0434a04da121995d81d7c5ded02cf2e70954dca49059485ea38310b7
         3b96fa1ec78619c5a1e524472778331fbe009fa156b5cdfc1bd66c5c
         9d7a05ab193dbe7b02dd8ba737887e41e9000b924368f33e4d88ffaf
         55b297d92ad0dee4ddb86d1992
```

B.6.3. Test vector for message from B

```
     Inputs
       ADb = b'ADb'
       yb (big endian): (length: 48 bytes)
         50b0e36b95a2edfaa8342b843dddc90b175330f2399c1b36586dedda
         3c255975f30be6a750f9404fccc62a6323b5e471
     Outputs
       Yb: (length: 97 bytes)
         040a6559147fd492ad74ab1f4def6196fd6399540e84706227a1f90d
         104cdaeb630b7c5c18748deb25653ad2a4cb5e6274841cad328bb031
         2628b9b1f51bea72b8c610999a6730f752649205ae85c452ef83f98b
         e715cd0103186874b07cf02074
       Alternative correct value for Yb: (-yb)*g:
       (length: 97 bytes)
         040a6559147fd492ad74ab1f4def6196fd6399540e84706227a1f90d
         104cdaeb630b7c5c18748deb25653ad2a4cb5e62747be352cd744fce
         d9d7464e0ae4158d4739ef666598cf08ad9b6dfa517a3bad0f7c0674
         17ea32fefce7978b50830fdf8b
```

B.6.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 48 bytes)
         a9acaea95692a64462067fe8e4321f2fca9793a8a0420f0e253ed0d6
         db858fe161de7576206a8a35bd4a60e00724fd3e
       scalar_mult_vfy(yb,Ya): (length: 48 bytes)
         a9acaea95692a64462067fe8e4321f2fca9793a8a0420f0e253ed0d6
         db858fe161de7576206a8a35bd4a60e00724fd3e
```

B.6.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 204 bytes)
  610434a04da121995d81d7c5ded02cf2e70954dca49059485ea38310
  b73b96fa1ec78619c5a1e524472778331fbe009fa1564a3203e42993
  a36285fa54e6c24184fd227458c87781be16fff46dbc970cc1b17700
  50a94d6826d52f211b234792e66d0341446161040a6559147fd492ad
  74ab1f4def6196fd6399540e84706227a1f90d104cdaeb630b7c5c18
  748deb25653ad2a4cb5e6274841cad328bb0312628b9b1f51bea72b8
  c610999a6730f752649205ae85c452ef83f98be715cd0103186874b0
  7cf0207403414462
DSI = G.DSI_ISK, b'CPaceP384_XMD:SHA-384_SSWU_NU__ISK':
(length: 34 bytes)
  4350616365503338345f584d443a5348412d3338345f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 305 bytes)
  2243506163655033383 45f584d443a5348412d3338345f535357555f
  4e555f5f49534b105b3773aa90e8f23c61563a4b645b276c30a9acae
  a95692a64462067fe8e4321f2fca9793a8a0420f0e253ed0d6db858f
  e161de7576206a8a35bd4a60e00724fd3e610434a04da121995d81d7
  c5ded02cf2e70954dca49059485ea38310b73b96fa1ec78619c5a1e5
  24472778331fbe009fa1564a3203e42993a36285fa54e6c24184fd22
  7458c87781be16fff46dbc970cc1b1770050a94d6826d52f211b2347
  92e66d0341446161040a6559147fd492ad74ab1f4def6196fd639954
  0e84706227a1f90d104cdaeb630b7c5c18748deb25653ad2a4cb5e62
  74841cad328bb0312628b9b1f51bea72b8c610999a6730f752649205
  ae85c452ef83f98be715cd0103186874b07cf0207403414462
ISK result: (length: 48 bytes)
  d1b74375c7d63d7de246cbf3fc2b3092645c73a0aa816989c0de6048
  ed4ece6a54df82d05d2be3498cb9288be7bdbdb9
```

B.6.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 206 bytes)
  6f63610434a04da121995d81d7c5ded02cf2e70954dca49059485ea3
  8310b73b96fa1ec78619c5a1e524472778331fbe009fa1564a3203e4
  2993a36285fa54e6c24184fd227458c87781be16fff46dbc970cc1b1
  770050a94d6826d52f211b234792e66d0341446161040a6559147fd4
  92ad74ab1f4def6196fd6399540e84706227a1f90d104cdaeb630b7c
  5c18748deb25653ad2a4cb5e6274841cad328bb0312628b9b1f51bea
  72b8c610999a6730f752649205ae85c452ef83f98be715cd01031868
  74b07cf0207403414462
DSI = G.DSI_ISK, b'CPaceP384_XMD:SHA-384_SSWU_NU__ISK':
(length: 34 bytes)
  4350616365503338345f584d443a5348412d3338345f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
(length: 307 bytes)
  2243506163655033383 45f584d443a5348412d3338345f535357555f
  4e555f5f49534b105b3773aa90e8f23c61563a4b645b276c30a9acae
  a95692a64462067fe8e4321f2fca9793a8a0420f0e253ed0d6db858f
  e161de7576206a8a35bd4a60e00724fd3e6f63610434a04da121995d
  81d7c5ded02cf2e70954dca49059485ea38310b73b96fa1ec78619c5
  a1e524472778331fbe009fa1564a3203e42993a36285fa54e6c24184
  fd227458c87781be16fff46dbc970cc1b1770050a94d6826d52f211b
  234792e66d0341446161040a6559147fd492ad74ab1f4def6196fd63
  99540e84706227a1f90d104cdaeb630b7c5c18748deb25653ad2a4cb
  5e6274841cad328bb0312628b9b1f51bea72b8c610999a6730f75264
  9205ae85c452ef83f98be715cd0103186874b07cf0207403414462
ISK result: (length: 48 bytes)
  a051d4532ca9fb6774e097ebac69c1d6a18144a15421dc155d0b1e8a
  ef9f9d8c0fe86e85d3cbee7796ff50171f42801b
```

B.6.7. Test vector for optional output of session id

```
H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
(length: 48 bytes)
  8d5a03946a69ffa12cd6efd469fe8671bbc25fad6db2656f3963c94e
  9c940bdecc2bd555474c211817787d5cf7870ed1
H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
(length: 48 bytes)
  c0729db25db40c48a35f787d5410b2cb9a2d9f5e9cf1cf159ed2f63c
  6b2185e597e176cc221422e9496eeda2bf123c8b
```

B.6.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
    0x5b,0x37,0x73,0xaa,0x90,0xe8,0xf2,0x3c,0x61,0x56,0x3a,0x4b,
    0x64,0x5b,0x27,0x6c,
   };
   const unsigned char tc_g[] = {
    0x04,0xff,0xe1,0xbd,0xc3,0x29,0x3f,0xdb,0xe3,0x1b,0x29,0x59,
    0x91,0x6e,0x52,0xc0,0x18,0xe9,0x23,0xea,0xc9,0x98,0x36,0xbd,
    0x9a,0x1c,0xbe,0xec,0x79,0x4a,0x8d,0x4d,0x78,0xba,0xa3,0x2c,
    0xda,0xfc,0x96,0x85,0xbc,0x10,0x67,0xa7,0x80,0xf4,0xad,0x9c,
    0x8a,0x6c,0x6e,0x16,0x4a,0xa4,0x29,0x06,0xd1,0xe2,0x7f,0x78,
    0x25,0x81,0xad,0xc8,0xe0,0x10,0x92,0x19,0x62,0x6a,0x2b,0x8f,
    0xbd,0xc3,0x46,0x02,0xe4,0x08,0x45,0x54,0xbd,0xd6,0xc0,0xc6,
    0x98,0xdd,0x65,0x7a,0xc8,0xe3,0x1b,0x2b,0xcc,0xe1,0xc7,0xb0,
    0xd8,
   };
   const unsigned char tc_ya[] = {
    0xef,0x43,0x3d,0xd5,0xad,0x14,0x2c,0x86,0x0e,0x7c,0xb6,0x40,
    0x0d,0xd3,0x15,0xd3,0x88,0xd5,0xec,0x54,0x20,0xc5,0x50,0xe9,
    0xd6,0xf0,0x90,0x7f,0x37,0x5d,0x98,0x8b,0xc4,0xd7,0x04,0x83,
    0x7e,0x43,0x56,0x1c,0x49,0x7e,0x7d,0xd9,0x3e,0xdc,0xdb,0x9d,
   };
   const unsigned char tc_ADa[] = {
    0x41,0x44,0x61,
   };
   const unsigned char tc_Ya[] = {
    0x04,0x34,0xa0,0x4d,0xa1,0x21,0x99,0x5d,0x81,0xd7,0xc5,0xde,
    0xd0,0x2c,0xf2,0xe7,0x09,0x54,0xdc,0xa4,0x90,0x59,0x48,0x5e,
    0xa3,0x83,0x10,0xb7,0x3b,0x96,0xfa,0x1e,0xc7,0x86,0x19,0xc5,
    0xa1,0xe5,0x24,0x47,0x27,0x78,0x33,0x1f,0xbe,0x00,0x9f,0xa1,
    0x56,0x4a,0x32,0x03,0xe4,0x29,0x93,0xa3,0x62,0x85,0xfa,0x54,
    0xe6,0xc2,0x41,0x84,0xfd,0x22,0x74,0x58,0xc8,0x77,0x81,0xbe,
    0x16,0xff,0xf4,0x6d,0xbc,0x97,0x0c,0xc1,0xb1,0x77,0x00,0x50,
    0xa9,0x4d,0x68,0x26,0xd5,0x2f,0x21,0x1b,0x23,0x47,0x92,0xe6,
    0x6d,
   };
   const unsigned char tc_yb[] = {
    0x50,0xb0,0xe3,0x6b,0x95,0xa2,0xed,0xfa,0xa8,0x34,0x2b,0x84,
    0x3d,0xdd,0xc9,0x0b,0x17,0x53,0x30,0xf2,0x39,0x9c,0x1b,0x36,
    0x58,0x6d,0xed,0xda,0x3c,0x25,0x59,0x75,0xf3,0x0b,0xe6,0xa7,
    0x50,0xf9,0x40,0x4f,0xcc,0xc6,0x2a,0x63,0x23,0xb5,0xe4,0x71,
```

```
   };
   const unsigned char tc_ADb[] = {
    0x41,0x44,0x62,
   };
   const unsigned char tc_Yb[] = {
    0x04,0x0a,0x65,0x59,0x14,0x7f,0xd4,0x92,0xad,0x74,0xab,0x1f,
    0x4d,0xef,0x61,0x96,0xfd,0x63,0x99,0x54,0x0e,0x84,0x70,0x62,
    0x27,0xa1,0xf9,0x0d,0x10,0x4c,0xda,0xeb,0x63,0x0b,0x7c,0x5c,
    0x18,0x74,0x8d,0xeb,0x25,0x65,0x3a,0xd2,0xa4,0xcb,0x5e,0x62,
    0x74,0x84,0x1c,0xad,0x32,0x8b,0xb0,0x31,0x26,0x28,0xb9,0xb1,
    0xf5,0x1b,0xea,0x72,0xb8,0xc6,0x10,0x99,0x9a,0x67,0x30,0xf7,
    0x52,0x64,0x92,0x05,0xae,0x85,0xc4,0x52,0xef,0x83,0xf9,0x8b,
    0xe7,0x15,0xcd,0x01,0x03,0x18,0x68,0x74,0xb0,0x7c,0xf0,0x20,
    0x74,
   };
   const unsigned char tc_K[] = {
    0xa9,0xac,0xae,0xa9,0x56,0x92,0xa6,0x44,0x62,0x06,0x7f,0xe8,
    0xe4,0x32,0x1f,0x2f,0xca,0x97,0x93,0xa8,0xa0,0x42,0x0f,0x0e,
    0x25,0x3e,0xd0,0xd6,0xdb,0x85,0x8f,0xe1,0x61,0xde,0x75,0x76,
    0x20,0x6a,0x8a,0x35,0xbd,0x4a,0x60,0xe0,0x07,0x24,0xfd,0x3e,
   };
   const unsigned char tc_ISK_IR[] = {
    0xd1,0xb7,0x43,0x75,0xc7,0xd6,0x3d,0x7d,0xe2,0x46,0xcb,0xf3,
    0xfc,0x2b,0x30,0x92,0x64,0x5c,0x73,0xa0,0xaa,0x81,0x69,0x89,
    0xc0,0xde,0x60,0x48,0xed,0x4e,0xce,0x6a,0x54,0xdf,0x82,0xd0,
    0x5d,0x2b,0xe3,0x49,0x8c,0xb9,0x28,0x8b,0xe7,0xbd,0xbd,0xb9,
   };
   const unsigned char tc_ISK_SY[] = {
    0xa0,0x51,0xd4,0x53,0x2c,0xa9,0xfb,0x67,0x74,0xe0,0x97,0xeb,
    0xac,0x69,0xc1,0xd6,0xa1,0x81,0x44,0xa1,0x54,0x21,0xdc,0x15,
    0x5d,0x0b,0x1e,0x8a,0xef,0x9f,0x9d,0x8c,0x0f,0xe8,0x6e,0x85,
    0xd3,0xcb,0xee,0x77,0x96,0xff,0x50,0x17,0x1f,0x42,0x80,0x1b,
   };
   const unsigned char tc_ISK_SY[] = {
    0xa0,0x51,0xd4,0x53,0x2c,0xa9,0xfb,0x67,0x74,0xe0,0x97,0xeb,
    0xac,0x69,0xc1,0xd6,0xa1,0x81,0x44,0xa1,0x54,0x21,0xdc,0x15,
    0x5d,0x0b,0x1e,0x8a,0xef,0x9f,0x9d,0x8c,0x0f,0xe8,0x6e,0x85,
    0xd3,0xcb,0xee,0x77,0x96,0xff,0x50,0x17,0x1f,0x42,0x80,0x1b,
   };
   const unsigned char tc_sid_out_ir[] = {
    0x8d,0x5a,0x03,0x94,0x6a,0x69,0xff,0xa1,0x2c,0xd6,0xef,0xd4,
    0x69,0xfe,0x86,0x71,0xbb,0xc2,0x5f,0xad,0x6d,0xb2,0x65,0x6f,
    0x39,0x63,0xc9,0x4e,0x9c,0x94,0x0b,0xde,0xcc,0x2b,0xd5,0x55,
    0x47,0x4c,0x21,0x18,0x17,0x78,0x7d,0x5c,0xf7,0x87,0x0e,0xd1,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xc0,0x72,0x9d,0xb2,0x5d,0xb4,0x0c,0x48,0xa3,0x5f,0x78,0x7d,
    0x54,0x10,0xb2,0xcb,0x9a,0x2d,0x9f,0x5e,0x9c,0xf1,0xcf,0x15,
```

```
    0x9e,0xd2,0xf6,0x3c,0x6b,0x21,0x85,0xe5,0x97,0xe1,0x76,0xcc,
    0x22,0x14,0x22,0xe9,0x49,0x6e,0xed,0xa2,0xbf,0x12,0x3c,0x8b,
   };
```

B.6.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
    #czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI1
    #QjM3NzNBQTkwRThGMjNDNjE1NjNBNEI2NDVCMjc2QyIsICJnIjogIjA0RkZFMUJEQz
    #MyOTNGREJFMzFCMjk1OTkxNkU1MkMwMThFOTIzRUFDOTk4MzZCRDlBMUNCRUVDNzk0
    #QThENEQ3OEJBQTMyQ0RBRkM5Njg1QkMxMDY3QTc4MEY0QUQ5QzhBNkM2RTE2NEFBND
    #I5MDZEMUUyN0Y3ODI1ODFBREM4RTAxMDkyMTk2MjZBMkI4RkJEQzM0NjAyRTQwODQ1
    #NTRCREQ2QzBDNjk4REQ2NTdBQzhFMzFCMkJDQ0UxQzdCMEQ4IiwgInlhIjogIkVGND
    #MzREQ1QUQxNDJDODYwRTdDQjY0MDBERDMxNUQzODhENUVDNTQyMEM1NTBFOUQ2RjA5
    #MDdGMzc1RDk4OEJDNEQ3MDQ4MzdFNDM1NjFDNDk3RTdERDkzRURDREI5RCIsICJBRG
    #EiOiAiNDE0NDYxIiwgIllhIjogIjA0MzRBMDREQTEyMTk5NUQ4MUQ3QzVERUQwMkNG
    #MkU3MDk1NERDQTQ5MDU5NDg1RUEzODMxMEI3M0I5NkZBMUVDNzg2MTlDNUExRTUyND
    #Q3Mjc3ODMzMUZCRTAwOUZBMTU2NEEzMjAzRTQyOTkzQTM2Mjg1RkE1NEU2QzI0MTg0
    #RkQyMjc0NThDODc3ODFCRTE2RkZGNDZEQkM5NzBDQzFCMTc3MDA1MEE5NEQ2ODI2RD
    #UyRjIxMUIyMzQ3OTJFNjZEIiwgInliIjogIjUwQjBFMzZCOTVBMkVERkFBODM0MkI4
    #NDNERERDOTBCMTc1MzMwRjIzOTlDMUIzNjU4NkRFRERBM0MyNTU5NzVGMzBCRTZBNz
    #UwRjk0MDRGQ0NDNjJBNjMyM0I1RTQ3MSIsICJBRGIiOiAiNDE0NDYyIiwgIlliIjog
    #IjA0MEE2NTU5MTQ3RkQ0OTJBRDc0QUIxRjRERUY2MTk2RkQ2Mzk5NTQwRTg0NzA2Mj
    #I3QTFGOTBEMTA0Q0RBRUI2MzBCN0M1QzE4NzQ4REVCMjU2NTNBRDJBNENCNUU2Mjc0
    #ODQxQ0FEMzI4QkIwMzEyNjI4QjlCMUY1MUJFQTcyQjhDNjEwOTk5QTY3MzBGNzUyNj
    #Q5MjA1QUU4NUM0NTJFRjgzRjk4QkU3MTVDRDAxMDMxODY4NzRCMDdDRjAyMDc0Iiwg
    #IksiOiAiQTlBQ0FFQTk1NjkyQTY0NDYyMDY3RkU4RTQzMjFGMkZDQTk3OTNBOEEwND
    #IwRjBFMjUzRUQwRDZEQjg1OEZFMTYxREU3NTc2MjA2QThBMzVCRDRBNjBFMDA3MjRG
    #RDNFIiwgIklTS19JUiI6ICJEMUI3NDM3NUM3RDYzRDdERTI0NkNCRjNGQzJCMzA5Mj
    #Y0NUM3M0EwQUE4MTY5ODlDMERFNjA0OEVENEVDRTZBNTRERjgyRDA1RDJCRTM0OThD
    #QjkyODhCRTdCREJEQjkiLCAiSVNLX1NZIjogIkEwNTFENDUzMkNBOUZCNjc3NEUwOT
    #dFQkFDNjlDMUQ2QTE4MTQ0QTE1NDIxREMxNTVEMEIxRThBRUY5RjlEOEMwRkU4NkU4
    #NUQzQ0JFRTc3OTZGRjUwMTcxRjQyODAxQiIsICJzaWRfb3V0cHV0X2lyIjogIjhENU
    #EwMzk0NkE2OUZGQTEyQ0Q2RUZENDY5RkU4NjcxQkJDMjVGQUQ2REIyNjU2RjM5NjND
    #OTRFOUM5NDBCREVDQzJCRDU1NTQ3NEMyMTE4MTc3ODdENUNGNzg3MEVEMSIsICJzaW
    #Rfb3V0cHV0X29jIjogIkMwNzI5REIyNURCNDBDNDhBMzVGNzg3RDU0MTBCMkNCOUEy
    #RDlGNUU5Q0YxQ0YxNTlFRDJGNjNDNkIyMTg1RTU5N0UxNzZDQzIyMTQyMkU5NDk2RU
    #VEQTJCRjEyM0M4QiJ9
```

B.6.10. Test case for scalar_mult_vfy with correct inputs

```
      s: (length: 48 bytes)
        6e8a99a5cdd408eae98e1b8aed286e7b12adbbdac7f2c628d9060ce9
        2ae0d90bd57a564fd3500fbcce3425dc94ba0ade
      X: (length: 97 bytes)
        045b4cd53c4506cc04ba4c44f2762d5d32c3e55df25b8baa5571b165
        7ad9576efea8259f0684de065a470585b4be876748c7797054f3defe
        f21b77f83d53bac57c89d52aa4d6dd5872bd281989b138359698009f
        8ac1f301538badcce9d9f4036e
      G.scalar_mult(s,X) (full coordinates): (length: 97 bytes)
        0465c28db05fd9f9a93651c5cc31eae49c4e5246b46489b8f6105873
        3173a033cda76c3e3ea5352b804e67fdbe2e334be8245dad5c8c993e
        63bacf0456478f29b71b6c859f13676f84ff150d2741f028f560584a
        0bdba19a63df62c08949c2fd6d
      G.scalar_mult_vfy(s,X) (only X-coordinate):
      (length: 48 bytes)
        65c28db05fd9f9a93651c5cc31eae49c4e5246b46489b8f610587331
        73a033cda76c3e3ea5352b804e67fdbe2e334be8
```

B.6.11. Invalid inputs for scalar_mult_vfy

For these test cases scalar_mult_vfy(y,.) MUST return the representation of the neutral element G.I. When including Y_i1 or Y_i2 in messages of A or B the protocol MUST abort.

```
      s: (length: 48 bytes)
        6e8a99a5cdd408eae98e1b8aed286e7b12adbbdac7f2c628d9060ce9
        2ae0d90bd57a564fd3500fbcce3425dc94ba0ade
      Y_i1: (length: 97 bytes)
        045b4cd53c4506cc04ba4c44f2762d5d32c3e55df25b8baa5571b165
        7ad9576efea8259f0684de065a470585b4be876748c7797054f3defe
        f21b77f83d53bac57c89d52aa4d6dd5872bd281989b138359698009f
        8ac1f301538badcce9d9f40302
      Y_i2: (length: 1 bytes)
        00
      G.scalar_mult_vfy(s,Y_i1) = G.scalar_mult_vfy(s,Y_i2) = G.I
```

B.6.11.1. Testvectors as JSON file encoded as BASE64

```
#eyJWYWxpZCI6IHsicyI6ICI2RThBOTlBNUNERDQwOEVBRTk4RTFCOEFFRDI4NkU3Qj
#EyQURCQkRBQzdGMkM2MjhEOTA2MENFOTJBRTBEOTBCRDU3QTU2NEZEMzUwMEZCQ0NF
#MzQyNURDOTRCQTBBREUiLCAiWCI6ICIwNDVCNENENTNDNDUwNkNDMDRCQTRDNDRGMj
#c2MkQ1RDMyQzNFNTVERjI1QjhCQUE1NTcxQjE2NTdBRDk1NzZFRkVBODI1OUYwNjg0
#REUwNjVBNDcwNTg1QjRCRTg3Njc0OEM3Nzk3MDU0RjNERUZFRjIxQjc3RjgzRDUzQk
#FDNTdDODlENTJBQTRENkRENTg3MkJEMjgxOTg5QjEzODM1OTY5ODAwOUY4QUMxRjMw
#MTUzOEJBRENDRTlEOUY0MDM2RSIsICJHLnNjYWxhcl9tdWx0KHMsWCkgKGZ1bGwgY2
#9vcmRpbmF0ZXMpIjogIjA0NjVDMjhEQjA1RkQ5RjlBOTM2NTFDNUNDMzFFQUU0OUM0
#RTUyNDZCNDY0ODlCOEY2MTA1ODczMzE3M0EwMzNDREE3NkMzRTNFQTUzNTJCODA0RT
#Y3RkRCRTJFMzM0QkU4MjQ1REFENUM4Qzk5M0U2M0JBQ0YwNDU2NDc4RjI5QjcxQjZD
#ODU5RjEzNjc2Rjg0RkYxNTBEMjc0MUYwMjhGNTYwNTg0QTBCREJBMTlBNjNERjYyQz
#A4OTQ5QzJGRDZEIiwgIkcuc2NhbGFyX211bHRfdmZ5KHMsWCkgKG9ubHkgWC1jb29y
#ZGluYXRlKSI6ICI2NUMyOERCMDVGRDlGOUE5MzY1MUM1Q0MzMUVBRTQ5QzRFNTI0Nk
#I0NjQ4OUI4RjYxMDU4NzMzMTczQTAzM0NEQTc2QzNFM0VBNTM1MkI4MDRFNjdGREJF
#MkUzMzRCRTgifSwgIkludmFsaWQgWTEiOiAiMDQ1QjRDRDUzQzQ1MDZDQzA0QkE0Qz
#Q0RjI3NjJENUQzMkMzRTU1REYyNUI4QkFBNTU3MUIxNjU3QUQ5NTc2RUZFQTgyNTlG
#MDY4NERFMDY1QTQ3MDU4NUI0QkU4NzY3NDhDNzc5NzA1NEYzREVGRUYyMUI3N0Y4M0
#Q1M0JBQzU3Qzg5RDUyQUE0RDZERDU4NzJCRDI4MTk4OUIxMzgzNTk2OTgwMDlGOEFD
#MUYzMDE1MzhCQURDQ0U5RDlGNDAzMDIiLCAiSW52YWxpZCBZMiI6ICIwMCJ9
```

B.7. Test vector for CPace using group NIST P-521 and hash SHA-512

B.7.1. Test vectors for calculate_generator with group NIST P-521

Inputs

```
H   = SHA-512 with input block size 128 bytes.
PRS = b'Password' ; ZPAD length: 87 ;
DSI = b'CPaceP521_XMD:SHA-512_SSWU_NU_'
DST = b'CPaceP521_XMD:SHA-512_SSWU_NU__DST'
CI = b'oc\x0bB_responder\x0bA_initiator'
CI = 6f630b425f726573706f6e6465720b415f696e69746961746f72
sid = 7e4b4791d6a8ef019b936c79fb7f2c57
```

Outputs

generator_string(PRS,G.DSI,CI,sid,H.s_in_bytes):
(length: 172 bytes)

```
1e435061636550353231 5f584d443a5348412d3531325f535357555f
4e555f0850617373776f726457000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000001a6f630b425f726573706f6e
6465720b415f696e69746961746f72107e4b4791d6a8ef019b936c79
fb7f2c57
```

generator g: (length: 133 bytes)

```
0400e58a8fbf08b38e34a3676f6d690bed58aa4115ff32a57ec87172
fc2a1fb89d03258c6429c464981b3284b5fedbd1244bf27432008868
7065b9075dd558e14ed69901d2162db1ba3a49c97dca7c902cb1b96b
abe21a31942114c860665b35c46b8213f6de17194de54c441083dd11
63d5907adad8824bb1307dcf6a55c11a8f01d9789b
```

B.7.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS01MTIiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
#E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiA4NywgIkRTSSI6ICI0MzUwNjE2
#MzY1NTAzNTMyMzE1RjU4NEQ0NDNBNTM0ODQxMkQzNTMxMzI1RjUzNTM1NzU1NUY0RT
#U1NUYiLCAiQ0kiOiAiNkY2MzBCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjBCNDE1RjY5
#NkU2OTc0Njk2MTc0NkY3MiIsICJzaWQiOiAiN0U0QjQ3OTFENkE4RUYwMTlCOTM2Qz
#c5RkI3RjJDNTciLCAiZ2VuZXJhdG9yX3N0cmluZyhHLkRTSSxQUlMsQ0ksc2lkLEgu
#c19pbl9ieXRlcykiOiAiMUU0MzUwNjE2MzY1NTAzNTMyMzE1RjU4NEQ0NDNBNTM0OD
#QxMkQzNTMxMzI1RjUzNTM1NzU1NUY0RTU1NUYwODUwNjE3MzczNzc2RjcyNjQ1NzAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDFBNkY2MzBCNDI1RjcyNjU3MzcwNkY2RTY0
#NjU3MjBCNDE1RjY5NkU2OTc0Njk2MTc0NkY3MjEwN0U0QjQ3OTFENkE4RUYwMTlCOT
#M2Qzc5RkI3RjJDNTciLCAiZ2VuZXJhdG9yIGciOiAiMDQwMEU1OEE4RkJGMDhCMzhF
#MzRBMzY3NkY2RDY5MEJFRDU4QUE0MTE1RkYzMkE1N0VDODcxNzJGQzJBMUZCODlEMD
#MyNThDNjQyOUM0NjQ5ODFCMzI4NEI1RkVEQkQxMjQ0QkYyNzQzMjAwODg2ODcwNjVC
#OTA3NURENTU4RTE0RUQ2OTkwMUQyMTYyREIxQkEzQTQ5Qzk3RENBN0M5MDJDQjFCOT
#ZCQUJFMjFBMzE5NDIxMTRDODYwNjY1QjM1QzQ2QjgyMTNGNkRFMTcxOTRERTU0QzQ0
#MTA4M0REMTE2M0Q1OTA3QURBRDg4MjRCQjEzMDdEQ0Y2QTU1QzExQThGMDFEOTc4OU
#IifQ==
```

B.7.2. Test vector for message from A

```
Inputs
  ADa = b’ADa’
  ya (big endian): (length: 66 bytes)
    006367e9c2aeff9f1db19af600cca73343d47cbe446cebbd1ccd783f
    82755a872da86fd0707eb3767c6114f1803deb62d63bdd1e613f67e6
    3e8c141ee5310e3ee819
Outputs
  Ya: (length: 133 bytes)
    0400c2bfd794467f4438277e85a42e101fa4061e1ef6e05f81e5381f
    30e73b341dd726089cb6a6bbe5a509fad009857488db7130ff768090
    7312eb724cddb4dcce675b0098ad400fef80e1deb4bc1756c43961ef
    60b85f2d62ed458454e11616a5d1df1e5809636821a73662f9f12254
    e6f9950dd01fa8e26a8b20736fb63c63c81094f681
  Alternative correct value for Ya: (-ya)*g:
  (length: 133 bytes)
    0400c2bfd794467f4438277e85a42e101fa4061e1ef6e05f81e5381f
    30e73b341dd726089cb6a6bbe5a509fad009857488db7130ff768090
    7312eb724cddb4dcce675b016752bff0107f1e214b43e8a93bc69e10
    9f47a0d29d12ba7bab1ee9e95a2e20e1a7f69c97de58c99d060eddab
    19066af22fe0571d9574df8c9049c39c37ef6b097e
```

B.7.3. Test vector for message from B

```
     Inputs
       ADb = b'ADb'
       yb (big endian): (length: 66 bytes)
         009227bf8dc741dacc9422f8bf3c0e96fce9587bc562eaafe0dc5f6f
         82f28594e4a6f98553560c62b75fa4abb198cecbbb86ebd41b0ea025
         4cde78ac68d39a240ae7
     Outputs
       Yb: (length: 133 bytes)
         0400706ea69b2b7167773248ea6e69a574e9dd2ff8a3d04a6e07f70c
         709869ca486827d59f9290599d1cf94e1a03fc242e2b1316afe2fa21
         8bfaeb3e1ffd9f19bf062d01f6b15c9c3651be4c08baf01eec25c818
         ee12c6edc4620644b1d97cf24f868732d56fe45ce78e302c221c92f4
         03e0fa3207de8bb41b388d81046a298ed8ddac9b2a
       Alternative correct value for Yb: (-yb)*g:
       (length: 133 bytes)
         0400706ea69b2b7167773248ea6e69a574e9dd2ff8a3d04a6e07f70c
         709869ca486827d59f9290599d1cf94e1a03fc242e2b1316afe2fa21
         8bfaeb3e1ffd9f19bf062d00094ea363c9ae41b3f7450fe113da37e7
         11ed39123b9df9bb4e26830db07978cd2a901ba31871cfd3dde36d0b
         fc1f05cdf821744be4c7727efb95d67127225364d5
```

B.7.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 66 bytes)
         018e0e7e9ade74917c11c0f6b52f95ed871eab235437cbee8b5c2509
         516e787a80e825ed5d539fa6a0ec32c48fa8fabe85809d000d0cfd30
         832c23d477c991bea8e5
       scalar_mult_vfy(yb,Ya): (length: 66 bytes)
         018e0e7e9ade74917c11c0f6b52f95ed871eab235437cbee8b5c2509
         516e787a80e825ed5d539fa6a0ec32c48fa8fabe85809d000d0cfd30
         832c23d477c991bea8e5
```

B.7.5. Test vector for ISK calculation initiator/responder

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 278 bytes)
  85010400c2bfd794467f4438277e85a42e101fa4061e1ef6e05f81e5
  381f30e73b341dd726089cb6a6bbe5a509fad009857488db7130ff76
  80907312eb724cddb4dcce675b0098ad400fef80e1deb4bc1756c439
  61ef60b85f2d62ed458454e11616a5d1df1e5809636821a73662f9f1
  2254e6f9950dd01fa8e26a8b20736fb63c63c81094f6810341446185
  010400706ea69b2b7167773248ea6e69a574e9dd2ff8a3d04a6e07f7
  0c709869ca486827d59f9290599d1cf94e1a03fc242e2b1316afe2fa
  218bfaeb3e1ffd9f19bf062d01f6b15c9c3651be4c08baf01eec25c8
  18ee12c6edc4620644b1d97cf24f868732d56fe45ce78e302c221c92
  f403e0fa3207de8bb41b388d81046a298ed8ddac9b2a03414462
DSI = G.DSI_ISK, b'CPaceP521_XMD:SHA-512_SSWU_NU__ISK':
(length: 34 bytes)
  43506163655035323 15f584d443a5348412d3531325f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 397 bytes)
  2243506163655035323 15f584d443a5348412d3531325f535357555f
  4e555f5f49534b107e4b4791d6a8ef019b936c79fb7f2c5742018e0e
  7e9ade74917c11c0f6b52f95ed871eab235437cbee8b5c2509516e78
  7a80e825ed5d539fa6a0ec32c48fa8fabe85809d000d0cfd30832c23
  d477c991bea8e585010400c2bfd794467f4438277e85a42e101fa406
  1e1ef6e05f81e5381f30e73b341dd726089cb6a6bbe5a509fad00985
  7488db7130ff7680907312eb724cddb4dcce675b0098ad400fef80e1
  deb4bc1756c43961ef60b85f2d62ed458454e11616a5d1df1e580963
  6821a73662f9f12254e6f9950dd01fa8e26a8b20736fb63c63c81094
  f6810341446185010400706ea69b2b7167773248ea6e69a574e9dd2f
  f8a3d04a6e07f70c709869ca486827d59f9290599d1cf94e1a03fc24
  2e2b1316afe2fa218bfaeb3e1ffd9f19bf062d01f6b15c9c3651be4c
  08baf01eec25c818ee12c6edc4620644b1d97cf24f868732d56fe45c
  e78e302c221c92f403e0fa3207de8bb41b388d81046a298ed8ddac9b
  2a03414462
ISK result: (length: 64 bytes)
  1669a0a29726adc7eea2510d6f7e004a135fa63ac3c9f9e6ce53cba5
  d5e3781aced515956041e43358409a13ef90ddc3c36fd8d7d81424c8
  e94592e21854260a
```

B.7.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 280 bytes)
  6f6385010400c2bfd794467f4438277e85a42e101fa4061e1ef6e05f
  81e5381f30e73b341dd726089cb6a6bbe5a509fad009857488db7130
  ff7680907312eb724cddb4dcce675b0098ad400fef80e1deb4bc1756
  c43961ef60b85f2d62ed458454e11616a5d1df1e5809636821a73662
  f9f12254e6f9950dd01fa8e26a8b20736fb63c63c81094f681034144
  6185010400706ea69b2b7167773248ea6e69a574e9dd2ff8a3d04a6e
  07f70c709869ca486827d59f9290599d1cf94e1a03fc242e2b1316af
  e2fa218bfaeb3e1ffd9f19bf062d01f6b15c9c3651be4c08baf01eec
  25c818ee12c6edc4620644b1d97cf24f868732d56fe45ce78e302c22
  1c92f403e0fa3207de8bb41b388d81046a298ed8ddac9b2a03414462
DSI = G.DSI_ISK, b'CPaceP521_XMD:SHA-512_SSWU_NU__ISK':
(length: 34 bytes)
  435061636550353231 5f584d443a5348412d3531325f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_oc(Ya,ADa,Yb,ADb):
(length: 399 bytes)
  2243506163655035323 15f584d443a5348412d3531325f535357555f
  4e555f5f49534b107e4b4791d6a8ef019b936c79fb7f2c5742018e0e
  7e9ade74917c11c0f6b52f95ed871eab235437cbee8b5c2509516e78
  7a80e825ed5d539fa6a0ec32c48fa8fabe85809d000d0cfd30832c23
  d477c991bea8e56f6385010400c2bfd794467f4438277e85a42e101f
  a4061e1ef6e05f81e5381f30e73b341dd726089cb6a6bbe5a509fad0
  09857488db7130ff7680907312eb724cddb4dcce675b0098ad400fef
  80e1deb4bc1756c43961ef60b85f2d62ed458454e11616a5d1df1e58
  09636821a73662f9f12254e6f9950dd01fa8e26a8b20736fb63c63c8
  1094f681034144618501040 0706ea69b2b7167773248ea6e69a574e9
  dd2ff8a3d04a6e07f70c709869ca486827d59f9290599d1cf94e1a03
  fc242e2b1316afe2fa218bfaeb3e1ffd9f19bf062d01f6b15c9c3651
  be4c08baf01eec25c818ee12c6edc4620644b1d97cf24f868732d56f
  e45ce78e302c221c92f403e0fa3207de8bb41b388d81046a298ed8dd
  ac9b2a03414462
ISK result: (length: 64 bytes)
  f2f3bd8cd442a4e16659b47a7b7a84f29be75893ed2e5f772d7a3c8b
  779eb0df937a4ec50a4f1ff01ebbaa97d80e090ea69b00a95200ed25
  8e48c6f7e9d8fbc2
```

B.7.7. Test vector for optional output of session id

```
      H.hash(b"CPaceSidOut" + transcript_ir(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        56cc3cd8be77cdc84c0d1906de1ffc8ef7cbb326a3f05267b6e8c634
        4e2781eb20ef725e84cb1bb45927435051b4e0fae78e975bf15099f9
        e38d755413eee2fd
      H.hash(b"CPaceSidOut" + transcript_oc(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        a46c7189ba6a36c4447741e057da39c885b7d59e08bd2df1852a5271
        f2a8a2e9b187ccd07325a3eede646adee0c06fe58da77f74177896b2
        1053c5d107de006d
```

B.7.8. Corresponding C programming language initializers

```
   const unsigned char tc_PRS[] = {
    0x50,0x61,0x73,0x73,0x77,0x6f,0x72,0x64,
   };
   const unsigned char tc_CI[] = {
    0x6f,0x63,0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,
    0x65,0x72,0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,
    0x6f,0x72,
   };
   const unsigned char tc_sid[] = {
    0x7e,0x4b,0x47,0x91,0xd6,0xa8,0xef,0x01,0x9b,0x93,0x6c,0x79,
    0xfb,0x7f,0x2c,0x57,
   };
   const unsigned char tc_g[] = {
    0x04,0x00,0xe5,0x8a,0x8f,0xbf,0x08,0xb3,0x8e,0x34,0xa3,0x67,
    0x6f,0x6d,0x69,0x0b,0xed,0x58,0xaa,0x41,0x15,0xff,0x32,0xa5,
    0x7e,0xc8,0x71,0x72,0xfc,0x2a,0x1f,0xb8,0x9d,0x03,0x25,0x8c,
    0x64,0x29,0xc4,0x64,0x98,0x1b,0x32,0x84,0xb5,0xfe,0xdb,0xd1,
    0x24,0x4b,0xf2,0x74,0x32,0x00,0x88,0x68,0x70,0x65,0xb9,0x07,
    0x5d,0xd5,0x58,0xe1,0x4e,0xd6,0x99,0x01,0xd2,0x16,0x2d,0xb1,
    0xba,0x3a,0x49,0xc9,0x7d,0xca,0x7c,0x90,0x2c,0xb1,0xb9,0x6b,
    0xab,0xe2,0x1a,0x31,0x94,0x21,0x14,0xc8,0x60,0x66,0x5b,0x35,
    0xc4,0x6b,0x82,0x13,0xf6,0xde,0x17,0x19,0x4d,0xe5,0x4c,0x44,
    0x10,0x83,0xdd,0x11,0x63,0xd5,0x90,0x7a,0xda,0xd8,0x82,0x4b,
    0xb1,0x30,0x7d,0xcf,0x6a,0x55,0xc1,0x1a,0x8f,0x01,0xd9,0x78,
    0x9b,
   };
   const unsigned char tc_ya[] = {
    0x00,0x63,0x67,0xe9,0xc2,0xae,0xff,0x9f,0x1d,0xb1,0x9a,0xf6,
    0x00,0xcc,0xa7,0x33,0x43,0xd4,0x7c,0xbe,0x44,0x6c,0xeb,0xbd,
    0x1c,0xcd,0x78,0x3f,0x82,0x75,0x5a,0x87,0x2d,0xa8,0x6f,0xd0,
    0x70,0x7e,0xb3,0x76,0x7c,0x61,0x14,0xf1,0x80,0x3d,0xeb,0x62,
    0xd6,0x3b,0xdd,0x1e,0x61,0x3f,0x67,0xe6,0x3e,0x8c,0x14,0x1e,
    0xe5,0x31,0x0e,0x3e,0xe8,0x19,
   };
   const unsigned char tc_ADa[] = {
```

```
    0x41,0x44,0x61,
   };
   const unsigned char tc_Ya[] = {
    0x04,0x00,0xc2,0xbf,0xd7,0x94,0x46,0x7f,0x44,0x38,0x27,0x7e,
    0x85,0xa4,0x2e,0x10,0x1f,0xa4,0x06,0x1e,0x1e,0xf6,0xe0,0x5f,
    0x81,0xe5,0x38,0x1f,0x30,0xe7,0x3b,0x34,0x1d,0xd7,0x26,0x08,
    0x9c,0xb6,0xa6,0xbb,0xe5,0xa5,0x09,0xfa,0xd0,0x09,0x85,0x74,
    0x88,0xdb,0x71,0x30,0xff,0x76,0x80,0x90,0x73,0x12,0xeb,0x72,
    0x4c,0xdd,0xb4,0xdc,0xce,0x67,0x5b,0x00,0x98,0xad,0x40,0x0f,
    0xef,0x80,0xe1,0xde,0xb4,0xbc,0x17,0x56,0xc4,0x39,0x61,0xef,
    0x60,0xb8,0x5f,0x2d,0x62,0xed,0x45,0x84,0x54,0xe1,0x16,0x16,
    0xa5,0xd1,0xdf,0x1e,0x58,0x09,0x63,0x68,0x21,0xa7,0x36,0x62,
    0xf9,0xf1,0x22,0x54,0xe6,0xf9,0x95,0x0d,0xd0,0x1f,0xa8,0xe2,
    0x6a,0x8b,0x20,0x73,0x6f,0xb6,0x3c,0x63,0xc8,0x10,0x94,0xf6,
    0x81,
   };
   const unsigned char tc_yb[] = {
    0x00,0x92,0x27,0xbf,0x8d,0xc7,0x41,0xda,0xcc,0x94,0x22,0xf8,
    0xbf,0x3c,0x0e,0x96,0xfc,0xe9,0x58,0x7b,0xc5,0x62,0xea,0xaf,
    0xe0,0xdc,0x5f,0x6f,0x82,0xf2,0x85,0x94,0xe4,0xa6,0xf9,0x85,
    0x53,0x56,0x0c,0x62,0xb7,0x5f,0xa4,0xab,0xb1,0x98,0xce,0xcb,
    0xbb,0x86,0xeb,0xd4,0x1b,0x0e,0xa0,0x25,0x4c,0xde,0x78,0xac,
    0x68,0xd3,0x9a,0x24,0x0a,0xe7,
   };
   const unsigned char tc_ADb[] = {
    0x41,0x44,0x62,
   };
   const unsigned char tc_Yb[] = {
    0x04,0x00,0x70,0x6e,0xa6,0x9b,0x2b,0x71,0x67,0x77,0x32,0x48,
    0xea,0x6e,0x69,0xa5,0x74,0xe9,0xdd,0x2f,0xf8,0xa3,0xd0,0x4a,
    0x6e,0x07,0xf7,0x0c,0x70,0x98,0x69,0xca,0x48,0x68,0x27,0xd5,
    0x9f,0x92,0x90,0x59,0x9d,0x1c,0xf9,0x4e,0x1a,0x03,0xfc,0x24,
    0x2e,0x2b,0x13,0x16,0xaf,0xe2,0xfa,0x21,0x8b,0xfa,0xeb,0x3e,
    0x1f,0xfd,0x9f,0x19,0xbf,0x06,0x2d,0x01,0xf6,0xb1,0x5c,0x9c,
    0x36,0x51,0xbe,0x4c,0x08,0xba,0xf0,0x1e,0xec,0x25,0xc8,0x18,
    0xee,0x12,0xc6,0xed,0xc4,0x62,0x06,0x44,0xb1,0xd9,0x7c,0xf2,
    0x4f,0x86,0x87,0x32,0xd5,0x6f,0xe4,0x5c,0xe7,0x8e,0x30,0x2c,
    0x22,0x1c,0x92,0xf4,0x03,0xe0,0xfa,0x32,0x07,0xde,0x8b,0xb4,
    0x1b,0x38,0x8d,0x81,0x04,0x6a,0x29,0x8e,0xd8,0xdd,0xac,0x9b,
    0x2a,
   };
   const unsigned char tc_K[] = {
    0x01,0x8e,0x0e,0x7e,0x9a,0xde,0x74,0x91,0x7c,0x11,0xc0,0xf6,
    0xb5,0x2f,0x95,0xed,0x87,0x1e,0xab,0x23,0x54,0x37,0xcb,0xee,
    0x8b,0x5c,0x25,0x09,0x51,0x6e,0x78,0x7a,0x80,0xe8,0x25,0xed,
    0x5d,0x53,0x9f,0xa6,0xa0,0xec,0x32,0xc4,0x8f,0xa8,0xfa,0xbe,
    0x85,0x80,0x9d,0x00,0x0d,0x0c,0xfd,0x30,0x83,0x2c,0x23,0xd4,
    0x77,0xc9,0x91,0xbe,0xa8,0xe5,
```

```
   };
   const unsigned char tc_ISK_IR[] = {
    0x16,0x69,0xa0,0xa2,0x97,0x26,0xad,0xc7,0xee,0xa2,0x51,0x0d,
    0x6f,0x7e,0x00,0x4a,0x13,0x5f,0xa6,0x3a,0xc3,0xc9,0xf9,0xe6,
    0xce,0x53,0xcb,0xa5,0xd5,0xe3,0x78,0x1a,0xce,0xd5,0x15,0x95,
    0x60,0x41,0xe4,0x33,0x58,0x40,0x9a,0x13,0xef,0x90,0xdd,0xc3,
    0xc3,0x6f,0xd8,0xd7,0xd8,0x14,0x24,0xc8,0xe9,0x45,0x92,0xe2,
    0x18,0x54,0x26,0x0a,
   };
   const unsigned char tc_ISK_SY[] = {
    0xf2,0xf3,0xbd,0x8c,0xd4,0x42,0xa4,0xe1,0x66,0x59,0xb4,0x7a,
    0x7b,0x7a,0x84,0xf2,0x9b,0xe7,0x58,0x93,0xed,0x2e,0x5f,0x77,
    0x2d,0x7a,0x3c,0x8b,0x77,0x9e,0xb0,0xdf,0x93,0x7a,0x4e,0xc5,
    0x0a,0x4f,0x1f,0xf0,0x1e,0xbb,0xaa,0x97,0xd8,0x0e,0x09,0x0e,
    0xa6,0x9b,0x00,0xa9,0x52,0x00,0xed,0x25,0x8e,0x48,0xc6,0xf7,
    0xe9,0xd8,0xfb,0xc2,
   };
   const unsigned char tc_ISK_SY[] = {
    0xf2,0xf3,0xbd,0x8c,0xd4,0x42,0xa4,0xe1,0x66,0x59,0xb4,0x7a,
    0x7b,0x7a,0x84,0xf2,0x9b,0xe7,0x58,0x93,0xed,0x2e,0x5f,0x77,
    0x2d,0x7a,0x3c,0x8b,0x77,0x9e,0xb0,0xdf,0x93,0x7a,0x4e,0xc5,
    0x0a,0x4f,0x1f,0xf0,0x1e,0xbb,0xaa,0x97,0xd8,0x0e,0x09,0x0e,
    0xa6,0x9b,0x00,0xa9,0x52,0x00,0xed,0x25,0x8e,0x48,0xc6,0xf7,
    0xe9,0xd8,0xfb,0xc2,
   };
   const unsigned char tc_sid_out_ir[] = {
    0x56,0xcc,0x3c,0xd8,0xbe,0x77,0xcd,0xc8,0x4c,0x0d,0x19,0x06,
    0xde,0x1f,0xfc,0x8e,0xf7,0xcb,0xb3,0x26,0xa3,0xf0,0x52,0x67,
    0xb6,0xe8,0xc6,0x34,0x4e,0x27,0x81,0xeb,0x20,0xef,0x72,0x5e,
    0x84,0xcb,0x1b,0xb4,0x59,0x27,0x43,0x50,0x51,0xb4,0xe0,0xfa,
    0xe7,0x8e,0x97,0x5b,0xf1,0x50,0x99,0xf9,0xe3,0x8d,0x75,0x54,
    0x13,0xee,0xe2,0xfd,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xa4,0x6c,0x71,0x89,0xba,0x6a,0x36,0xc4,0x44,0x77,0x41,0xe0,
    0x57,0xda,0x39,0xc8,0x85,0xb7,0xd5,0x9e,0x08,0xbd,0x2d,0xf1,
    0x85,0x2a,0x52,0x71,0xf2,0xa8,0xa2,0xe9,0xb1,0x87,0xcc,0xd0,
    0x73,0x25,0xa3,0xee,0xde,0x64,0x6a,0xde,0xe0,0xc0,0x6f,0xe5,
    0x8d,0xa7,0x7f,0x74,0x17,0x78,0x96,0xb2,0x10,0x53,0xc5,0xd1,
    0x07,0xde,0x00,0x6d,
   };
```

B.7.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICI2RjYzMEI0MjVGNzI2NT
    #czNzA2RjZFNjQ2NTcyMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyIiwgInNpZCI6ICI3
    #RTRCNDc5MUQ2QThFRjAxOUI5MzZDNzlGQjdGMkM1NyIsICJnIjogIjA0MDBFNThBOE
    #ZCRjA4QjM4RTM0QTM2NzZGNkQ2OTBCRUQ1OEFBNDExNUZGMzJBNTdFQzg3MTcyRkMy
    #QTFGQjg5RDAzMjU4QzY0MjlDNDY0OTgxQjMyODRCNUZFREJEMTI0NEJGMjc0MzIwMD
    #g4Njg3MDY1QjkwNzVERDU1OEUxNEVENjk5MDFEMjE2MkRCMUJBM0E0OUM5N0RDQTdD
    #OTAyQ0IxQjk2QkFCRTIxQTMxOTQyMTE0Qzg2MDY2NUIzNUM0NkI4MjEzRjZERTE3MT
    #k0REU1NEM0NDEwODNERDExNjNENTkwN0FEQUQ4ODI0QkIxMzA3RENGNkE1NUMxMUE4
    #RjAxRDk3ODlCIiwgInlhIjogIjAwNjM2N0U5QzJBRUZGOUYxREIxOUFGNjAwQ0NBNz
    #MzNDNENDdDQkU0NDZDRUJCRDFDQ0Q3ODNGODI3NTVBODcyREE4NkZEMDcwN0VCMzc2
    #N0M2MTE0RjE4MDNERUI2MkQ2M0JERDFFNjEzRjY3RTYzRThDMTQxRUU1MzEwRTNFRT
    #gxOSIsICJBRGEiOiAiNDE0NDYxIiwgIllhIjogIjA0MDBDMkJGRDc5NDQ2N0Y0NDM4
    #Mjc3RTg1QTQyRTEwMUZBNDA2MUUxRUY2RTA1RjgxRTUzODFGMzBFNzNCMzQxREQ3Mj
    #YwODlDQjZBNkJCRTVBNTA5RkFEMDA5ODU3NDg4REI3MTMwRkY3NjgwOTA3MzEyRUI3
    #MjRDRERCNERDQ0U2NzVCMDA5OEFENDAwRkVGODBFMURFQjRCQzE3NTZDNDM5NjFFRj
    #YwQjg1RjJENjJFRDQ1ODQ1NEUxMTYxNkE1RDFERjFFNTgwOTYzNjgyMUE3MzY2MkY5
    #RjEyMjU0RTZGOTk1MEREMDFGQThFMjZBOEIyMDczNkZCNjNDNjNDODEwOTRGNjgxIi
    #wgInliIjogIjAwOTIyN0JGOERDNzQxREFDQzk0MjJGOEJGM0MwRTk2RkNFOTU4N0JD
    #NTYyRUFBRkUwREM1RjZGODJGMjg1OTRFNEE2Rjk4NTUzNTYwQzYyQjc1RkE0QUJCMT
    #k4Q0VDQkJCODZFQkQ0MUIwRUEwMjU0Q0RFNzhBQzY4RDM5QTI0MEFFNyIsICJBRGIi
    #OiAiNDE0NDYyIiwgIlliIjogIjA0MDA3MDZFQTY5QjJCNzE2Nzc3MzI0OEVBNkU2OU
    #E1NzRFOUREMkZGOEEzRDA0QTZFMDdGNzBDNzA5ODY5Q0E0ODY4MjdENTlGOTI5MDU5
    #OUQxQ0Y5NEUxQTAzRkMyNDJFMkIxMzE2QUZFMkZBMjE4QkZBRUIzRTFGRkQ5RjE5Qk
    #YwNjJEMDFGNkIxNUM5QzM2NTFCRTRDMDhCQUYwMUVFQzI1QzgxOEVFMTJDNkVEQzQ2
    #MjA2NDRCMUQ5N0NGMjRGODY4NzMyRDU2RkU0NUNFNzhFMzAyQzIyMUM5MkY0MDNFME
    #ZBMzIwN0RFOEJCNDFCMzg4RDgxMDQ2QTI5OEVEOEREQUM5QjJBIiwgIksiOiAiMDE4
    #RTBFN0U5QURFNzQ5MTdDMTFDMEY2QjUyRjk1RUQ4NzFFQUIyMzU0MzdDQkVFOEI1Qz
    #I1MDk1MTZFNzg3QTgwRTgyNUVENUQ1MzlGQTZBMEVDMzJDNDhGQThGQUJFODU4MDlE
    #MDAwRDBDRkQzMDgzMkMyM0Q0NzdDOTkxQkVBOEU1IiwgIklTS19JUiI6ICIxNjY5QT
    #BBMjk3MjZBREM3RUVBMjUxMEQ2RjdFMDA0QTEzNUZBNjNBQzNDOUY5RTZDRTUzQ0JB
    #NUQ1RTM3ODFBQ0VENTE1OTU2MDQxRTQzMzU4NDA5QTEzRUY5MEREQzNDMzZGRDhEN0
    #Q4MTQyNEM4RTk0NTkyRTIxODU0MjYwQSIsICJJU0tfU1kiOiAiRjJGM0JEOENENDQy
    #QTRFMTY2NTlCNDdBN0I3QTg0RjI5QkU3NTg5M0VEMkU1Rjc3MkQ3QTNDOEI3NzlFQj
    #BERjkzN0E0RUM1MEE0RjFGRjAxRUJCQUE5N0Q4MEUwOTBFQTY5QjAwQTk1MjAwRUQy
    #NThFNDhDNkY3RTlEOEZCQzIiLCAic2lkX291dHB1dF9pciI6ICI1NkNDM0NEOEJFNz
    #dDREM4NEMwRDE5MDZERTFGRkM4RUY3Q0JCMzI2QTNGMDUyNjdCNkU4QzYzNDRFMjc4
    #MUVCMjBFRjcyNUU4NENCMUJCNDU5Mjc0MzUwNTFCNEUwRkFFNzhFOTc1QkYxNTA5OU
    #Y5RTM4RDc1NTQxM0VFRTJGRCIsICJzaWRfb3V0cHV0X29jIjogIkE0NkM3MTg5QkE2
    #QTM2QzQ0NDc3NDFFMDU3REEzOUM4ODVCN0Q1OUUwOEJEMkRGMTg1MkE1MjcxRjJBOE
    #EyRTlCMTg3Q0NEMDczMjVBM0VFREU2NDZBREVFMEMwNkZFNThEQTc3Rjc0MTc3ODk2
    #QjIxMDUzQzVEMTA3REUwMDZEIn0=
```

B.7.10. Test case for scalar_mult_vfy with correct inputs

```
      s: (length: 66 bytes)
        0182dd7925f1753419e4bf83429763acd37d64000cd5a175edf53a15
        87dd986bc95acc1506991702b6ba1a9ee2458fee8efc00198cf0088c
        480965ef65ff2048b856
      X: (length: 133 bytes)
        0400dc5078b24c4af1620cc10fbecc6cd8cf1cab0b011efb73c782f2
        26dc21c7ca7eb406be74a69ecba5b4a87c07cfc6e687b4beca9a6eda
        c95940a3b4120573b26a80005e697833b0ba285fce7b3f1f25243008
        860b8f1de710a0dcc05b0d20341efe90eb2bcca26797c2d85ae6ca74
        c00696cb1b13e40bda15b27964d7670576647bfab9
      G.scalar_mult(s,X) (full coordinates): (length: 133 bytes)
        040122f88ce73ec5aa2d1c8c5d04148760c3d97ba87daa10d8cb8bb7
        c73cf6e951fc922721bf1437995cfb13e132a78beb86389e60d3517c
        df6d99a8a2d6db19ef27bd0055af9e8ddcf337ce0a7c22a9c8099bc4
        a44faeded1eb72effd26e4f322217b67d60b944b267b3df5046078fd
        577f1785728f49b241fd5e8c83223a994a2d219281
      G.scalar_mult_vfy(s,X) (only X-coordinate):
      (length: 66 bytes)
        0122f88ce73ec5aa2d1c8c5d04148760c3d97ba87daa10d8cb8bb7c7
        3cf6e951fc922721bf1437995cfb13e132a78beb86389e60d3517cdf
        6d99a8a2d6db19ef27bd
```

B.7.11. Invalid inputs for scalar_mult_vfy

For these test cases scalar_mult_vfy(y,.) MUST return the representation of the neutral element G.I. When including Y_i1 or Y_i2 in messages of A or B the protocol MUST abort.

```
      s: (length: 66 bytes)
        0182dd7925f1753419e4bf83429763acd37d64000cd5a175edf53a15
        87dd986bc95acc1506991702b6ba1a9ee2458fee8efc00198cf0088c
        480965ef65ff2048b856
      Y_i1: (length: 133 bytes)
        0400dc5078b24c4af1620cc10fbecc6cd8cf1cab0b011efb73c782f2
        26dc21c7ca7eb406be74a69ecba5b4a87c07cfc6e687b4beca9a6eda
        c95940a3b4120573b26a80005e697833b0ba285fce7b3f1f25243008
        860b8f1de710a0dcc05b0d20341efe90eb2bcca26797c2d85ae6ca74
        c00696cb1b13e40bda15b27964d7670576647bfaf9
      Y_i2: (length: 1 bytes)
        00
      G.scalar_mult_vfy(s,Y_i1) = G.scalar_mult_vfy(s,Y_i2) = G.I
```

B.7.11.1. Testvectors as JSON file encoded as BASE64

#eyJWYWxpZCI6IHsicyI6ICIwMTgyREQ3OTI1RjE3NTM0MTlFNEJGODM0Mjk3NjNBQ0
#QzN0Q2NDAwMENENUExNzVFREY1M0ExNTg3REQ5ODZCQzk1QUNDMTUwNjk5MTcwMkI2
#QkExQTlFRTI0NThGRUU4RUZDMDAxOThDRjAwODhDNDgwOTY1RUY2NUZGMjA0OEI4NT
#YiLCAiWCI6ICIwNDAwREM1MDc4QjI0QzRBRjE2MjBDQzEwRkJFQ0M2Q0Q4Q0YxQ0FC
#MEIwMTFFRkI3M0M3ODJGMjI2REMyMUM3Q0E3RUI0MDZCRTc0QTY5RUNCQTVCNEE4N0
#MwN0NGQzZFNjg3QjRCRUNBOUE2RURBQzk1OTQwQTNCNDEyMDU3M0IyNkE4MDAwNUU2
#OTc4MzNCMEJBMjg1RkNFN0IzRjFGMjUyNDMwMDg4NjBCOEYxREU3MTBBMERDQzA1Qj
#BEMjAzNDFFRkU5MEVCMkJDQ0EyNjc5N0MyRDg1QUU2Q0E3NEMwMDY5NkNCMUIxM0U0
#MEJEQTE1QjI3OTY0RDc2NzA1NzY2NDdCRkFCOSIsICJHLnNjYWxhcl9tdWx0KHMsWC
#kgKGZ1bGwgY29vcmRpbmF0ZXMpIjogIjA0MDEyMkY4OENFNzNFQzVBQTJEMUM4QzVE
#MDQxNDg3NjBDM0Q5N0JBODdEQUExMEQ4Q0I4QkI3QzczQ0Y2RTk1MUZDOTIyNzIxQk
#YxNDM3OTk1Q0ZCMTNFMTMyQTc4QkVCODYzODlFNjBEMzUxN0NERjZEOTlBOEEyRDZE
#QjE5RUYyN0JEMDA1NUFGOUU4RERDRjMzN0NFMEE3QzIyQTlDODA5OUJDNEE0NEZBRU
#RFRDFFQjcyRUZGRDI2RTRGMzIyMjE3QjY3RDYwQjk0NEIyNjdCM0RGNTA0NjA3OEZE
#NTc3RjE3ODU3MjhGNDlCMjQxRkQ1RThDODMyMjNBOTk0QTJEMjE5MjgxIiwgIkcuc2
#NhbGFyX211bHRfdmZ5KHMsWCkgKG9ubHkgWC1jb29yZGluYXRlKSI6ICIwMTIyRjg4
#Q0U3M0VDNUFBMkQxQzhDNUQwNDE0ODc2MEMzRDk3QkE4N0RBQTEwRDhDQjhCQjdDNz
#NDRjZFOTUxRkM5MjI3MjFCRjE0Mzc5OTVDRkIxM0UxMzJBNzhCRUI4NjM4OUU2MEQz
#NTE3Q0RGNkQ5OUE4QTJENkRCMTlFRjI3QkQifSwgIkludmFsaWQgWTEiOiAiMDQwME
#RDNTA3OEIyNEM0QUYxNjIwQ0MxMEZCRUNDNkNEOENGMUNBQjBCMDExRUZCNzNDNzgy
#RjIyNkRDMjFDN0NBN0VCNDA2QkU3NEE2OUVDQkE1QjRBODdDMDdDRkM2RTY4N0I0Qk
#VDQTlBNkVEQUM5NTk0MEEzQjQxMjA1NzNCMjZBODAwMDVFNjk3ODMzQjBCQTI4NUZD
#RTdCM0YxRjI1MjQzMDA4ODYwQjhGMURFNzEwQTBEQ0MwNUIwRDIwMzQxRUZFOTBFQj
#JCQ0NBMjY3OTdDMkQ4NUFFNkNBNzRDMDA2OTZDQjFCMTNFNDBCREExNUIyNzk2NEQ3
#NjcwNTc2NjQ3QkZBRjkiLCAiSW52YWxpZCBZMiI6ICIwMCJ9

Authors' Addresses

Michel Abdalla
Nexus - San Francisco
Email: michel.abdalla@gmail.com

Bjoern Haase
Endress + Hauser Liquid Analysis - Gerlingen
Email: bjoern.m.haase@web.de

Julia Hesse
IBM Research Europe - Zurich
Email: juliahesse2@gmail.com