CFRG
Internet-Draft
Intended status: Informational
Expires: 24 September 2026

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

23 March 2026

CPace, a balanced composable PAKE
draft-irtf-cfrg-cpace-20

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

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

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

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

This Internet-Draft will expire on 24 September 2026.

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 [RFC9807].

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 +.

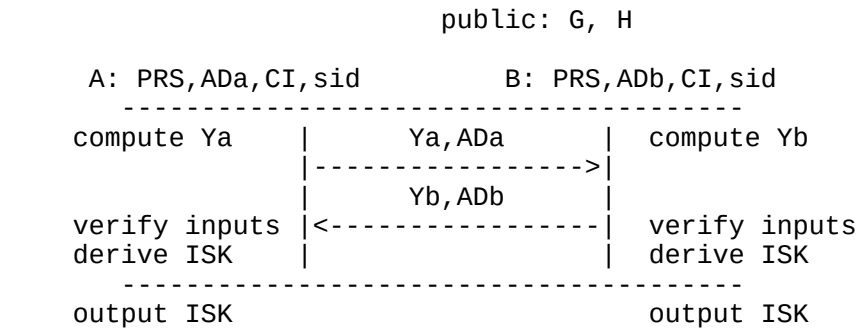
6. The CPace protocol

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 [RFC9496]. 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

[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>.

[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>.

[RFC9496] de Valence, H., Grigg, J., Hamburg, M., Lovecruft, I., Tankersley, G., and F. Valsorda, "The ristretto255 and decaf448 Groups", RFC 9496, DOI 10.17487/RFC9496, December 2023, <https://www.rfc-editor.org/rfc/rfc9496>.

[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>.

[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>.

[RFC9807] Bourdrez, D., Krawczyk, H., Lewi, K., and C. A. Wood, "The OPAQUE Augmented Password-Authenticated Key Exchange (aPAKE) Protocol", RFC 9807, DOI 10.17487/RFC9807, July 2025, <https://www.rfc-editor.org/rfc/rfc9807>.

[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
      727374757677787 97a7b7c7d7e7f
```

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'\x0bA_initiator\x0bB_responder'
  CI = 0b415f696e69746961746f720b425f726573706f6e646572
  sid = 7e4b4791d6a8ef019b936c79fb7f2c57
Outputs
  generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
  (length: 170 bytes)
    0843506163653235350850617373776f72646d000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    000000000000000000000000000000180b415f696e69746961746f
    720b425f726573706f6e646572107e4b4791d6a8ef019b936c79fb7f
    2c57
  hash generator string: (length: 32 bytes)
    03998087bdb1a2617bbe25ef5a7c18cd4f84f902328701790958755e
    e4aed1d3
  decoded field element of 255 bits: (length: 32 bytes)
    03998087bdb1a2617bbe25ef5a7c18cd4f84f902328701790958755e
    e4aed153
  generator g: (length: 32 bytes)
    d04bf6d41f6a289632a2e929fa29bebd51092512a7829fdde7d314b6
    2f05a73f
```

B.1.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS01MTIiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
#E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiAxMDksICJEU0kiOiAiNDM1MDYx
#NjM2NTMyMzUzNSIsICJDSSI6ICIwQjQxNUY2OTZFNjk3NDY5NjE3NDZGNzIwQjQyNU
#Y3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjdFNEI0NzkxRDZBOEVGMDE5Qjkz
#NkM3OUZCN0YyQzU3IiwgImdlbmVyYXRvcl9zdHJpbmcoRy5EU0ksUFJTLENJLHNpZC
#xILnNfaW5fYnl0ZXMpIjogIjA4NDM1MDYxNjM2NTMyMzUzNTA4NTA2MTczNzM3NzZG
#NzI2NDZEMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAxODBCNDE1RjY5NkU2OTc0Njk2MTc0Nk
#Y3MjBCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjEwN0U0QjQ3OTFENkE4RUYwMTlCOTM2
#Qzc5RkI3RjJDNTciLCAiaGFzaCBnZW5lcmF0b3Igc3RyaW5nIjogIjAzOTk4MDg3Qk
#RCMUEyNjE3QkJFMjVFRjVBN0MxOENENEY4NEY5MDIzMjg3MDE3OTA5NTg3NTVFRTRB
#RUQxRDMiLCAiZGVjb2RlZCBmaWVsZCBlbGVtZW50IG9mIDI1NSBiaXRzIjogIjAzOT
#k4MDg3QkRCMUEyNjE3QkJFMjVFRjVBN0MxOENENEY4NEY5MDIzMjg3MDE3OTA5NTg3
#NTVFRTRBRUQxNTMiLCAiZ2VuZXJhdG9yIGciOiAiRDA0QkY2RDQxRjZBMjg5NjMyQT
#JFOTI5RkEyOUJFQkQ1MTA5MjUxMkE3ODI5RkRERTdEMzE0QjYyRjA1QTczRiJ9
```

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)
    1d13c89278cdadd826f6d8d7f887701430f8380ddc17611cdd6dc989
    ce0c9f32
```

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)
    248cccf6d5cdc3646f0ad593f9e6cef4e69d4945f8372e623512ecea
    32185623
```

B.1.4. Test vector for secret points K

```
  scalar_mult_vfy(ya,Yb): (length: 32 bytes)
    5b067effbdc0b2a0e1d907b21ebb25cfedb96a852179a847c37e43ee
    71322c6b
  scalar_mult_vfy(yb,Ya): (length: 32 bytes)
    5b067effbdc0b2a0e1d907b21ebb25cfedb96a852179a847c37e43ee
    71322c6b
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 74 bytes)
  201d13c89278cdadd826f6d8d7f887701430f8380ddc17611cdd6dc9
  89ce0c9f320341446120248cccf6d5cdc3646f0ad593f9e6cef4e69d
  4945f8372e623512ecea3218562303414462
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
  2c57205b067effbdc0b2a0e1d907b21ebb25cfedb96a852179a847c3
  7e43ee71322c6b201d13c89278cdadd826f6d8d7f887701430f8380d
  dc17611cdd6dc989ce0c9f320341446120248cccf6d5cdc3646f0ad5
  93f9e6cef4e69d4945f8372e623512ecea3218562303414462
ISK result: (length: 64 bytes)
  6e19b875f7a561d6b3ca3dbb9ef42ac55de3e717881018204b8922b4
  d5e53bb2aa82c300bea7b65d2b671da71922ddf6472301b79bc270ad
  fa8bf413285f2263
```

B.1.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 76 bytes)
  6f6320248cccf6d5cdc3646f0ad593f9e6cef4e69d4945f8372e6235
  12ecea321856230341446220 1d13c89278cdadd826f6d8d7f8877014
  30f8380ddc17611cdd6dc989ce0c9f3203414461
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
  2c57205b067effbdc0b2a0e1d907b21ebb25cfedb96a852179a847c3
  7e43ee71322c6b6f6320248cccf6d5cdc3646f0ad593f9e6cef4e69d
  4945f8372e623512ecea321856230341446220 1d13c89278cdadd826
  f6d8d7f887701430f8380ddc17611cdd6dc989ce0c9f3203414461
ISK result: (length: 64 bytes)
  eef745e2f6e7ae2b1a1e53da340e777167a07fe150436648c51fb199
  c11f3cbabfc683a2b48e1af5881940dc398d375c95e6b4ae9948a45b
  8770de0656382be4
```

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

```
      H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        cbc73f62589bbc96ab6a95ec2363df621e93bc3b0cea83ba6b9571d0
        05fa8f5d2d08f7165622777fa484c02a9e6b20a84ee2dbebae8c53be
        757dcfc0eebdeb5f
      H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        3a504e9c7f1f7fa7314861e2c487d13f28566f3043f0ca760d22c491
        1aca0dd8b1f12a7ad0862eb92d08a76120140412ae6b8322e99d75cf
        1d20d8cfde2b40fe
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
    0xd0,0x4b,0xf6,0xd4,0x1f,0x6a,0x28,0x96,0x32,0xa2,0xe9,0x29,
    0xfa,0x29,0xbe,0xbd,0x51,0x09,0x25,0x12,0xa7,0x82,0x9f,0xdd,
    0xe7,0xd3,0x14,0xb6,0x2f,0x05,0xa7,0x3f,
   };
   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[] = {
    0x1d,0x13,0xc8,0x92,0x78,0xcd,0xad,0xd8,0x26,0xf6,0xd8,0xd7,
    0xf8,0x87,0x70,0x14,0x30,0xf8,0x38,0x0d,0xdc,0x17,0x61,0x1c,
    0xdd,0x6d,0xc9,0x89,0xce,0x0c,0x9f,0x32,
   };
   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[] = {
    0x24,0x8c,0xcc,0xf6,0xd5,0xcd,0xc3,0x64,0x6f,0x0a,0xd5,0x93,
    0xf9,0xe6,0xce,0xf4,0xe6,0x9d,0x49,0x45,0xf8,0x37,0x2e,0x62,
    0x35,0x12,0xec,0xea,0x32,0x18,0x56,0x23,
   };
   const unsigned char tc_K[] = {
    0x5b,0x06,0x7e,0xff,0xbd,0xc0,0xb2,0xa0,0xe1,0xd9,0x07,0xb2,
    0x1e,0xbb,0x25,0xcf,0xed,0xb9,0x6a,0x85,0x21,0x79,0xa8,0x47,
    0xc3,0x7e,0x43,0xee,0x71,0x32,0x2c,0x6b,
   };
   const unsigned char tc_ISK_IR[] = {
    0x6e,0x19,0xb8,0x75,0xf7,0xa5,0x61,0xd6,0xb3,0xca,0x3d,0xbb,
    0x9e,0xf4,0x2a,0xc5,0x5d,0xe3,0xe7,0x17,0x88,0x10,0x18,0x20,
    0x4b,0x89,0x22,0xb4,0xd5,0xe5,0x3b,0xb2,0xaa,0x82,0xc3,0x00,
    0xbe,0xa7,0xb6,0x5d,0x2b,0x67,0x1d,0xa7,0x19,0x22,0xdd,0xf6,
    0x47,0x23,0x01,0xb7,0x9b,0xc2,0x70,0xad,0xfa,0x8b,0xf4,0x13,
    0x28,0x5f,0x22,0x63,
   };
   const unsigned char tc_ISK_SY[] = {
    0xee,0xf7,0x45,0xe2,0xf6,0xe7,0xae,0x2b,0x1a,0x1e,0x53,0xda,
    0x34,0x0e,0x77,0x71,0x67,0xa0,0x7f,0xe1,0x50,0x43,0x66,0x48,
    0xc5,0x1f,0xb1,0x99,0xc1,0x1f,0x3c,0xba,0xbf,0xc6,0x83,0xa2,
    0xb4,0x8e,0x1a,0xf5,0x88,0x19,0x40,0xdc,0x39,0x8d,0x37,0x5c,
    0x95,0xe6,0xb4,0xae,0x99,0x48,0xa4,0x5b,0x87,0x70,0xde,0x06,
    0x56,0x38,0x2b,0xe4,
   };
   const unsigned char tc_sid_out_ir[] = {
    0xcb,0xc7,0x3f,0x62,0x58,0x9b,0xbc,0x96,0xab,0x6a,0x95,0xec,
    0x23,0x63,0xdf,0x62,0x1e,0x93,0xbc,0x3b,0x0c,0xea,0x83,0xba,
    0x6b,0x95,0x71,0xd0,0x05,0xfa,0x8f,0x5d,0x2d,0x08,0xf7,0x16,
    0x56,0x22,0x77,0x7f,0xa4,0x84,0xc0,0x2a,0x9e,0x6b,0x20,0xa8,
    0x4e,0xe2,0xdb,0xeb,0xae,0x8c,0x53,0xbe,0x75,0x7d,0xcf,0xc0,
    0xee,0xbd,0xeb,0x5f,
   };
   const unsigned char tc_sid_out_oc[] = {
    0x3a,0x50,0x4e,0x9c,0x7f,0x1f,0x7f,0xa7,0x31,0x48,0x61,0xe2,
    0xc4,0x87,0xd1,0x3f,0x28,0x56,0x6f,0x30,0x43,0xf0,0xca,0x76,
    0x0d,0x22,0xc4,0x91,0x1a,0xca,0x0d,0xd8,0xb1,0xf1,0x2a,0x7a,
    0xd0,0x86,0x2e,0xb9,0x2d,0x08,0xa7,0x61,0x20,0x14,0x04,0x12,
    0xae,0x6b,0x83,0x22,0xe9,0x9d,0x75,0xcf,0x1d,0x20,0xd8,0xcf,
    0xde,0x2b,0x40,0xfe,
   };
```

B.1.9. Testvectors as JSON file encoded as BASE64

```
#eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
#Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjdFNEI0
#NzkxRDZBOEVGMDE5QjkzNkM3OUZCN0YyQzU3IiwgImciOiAiRDA0QkY2RDQxRjZBMj
#g5NjMyQTJFOTI5RkEyOUJFQkQ1MTA5MjUxMkE3ODI5RkRERTdEMzE0QjYyRjA1QTcz
#RiIsICJ5YSI6ICIyMUI0RjRCRDlFNjRFRDM1NUMzRUI2NzZBMjhFQkVEQUY2RDhGMT
#dCREMzNjU5OTVCMzE5MDk3MTUzMDQ0MDgwIiwgIkFEYSI6ICI0MTQ0NjEiLCAiWWEi
#OiAiMUQxM0M4OTI3OENEQUREODI2RjZEOEQ3Rjg4NzcwMTQzMEY4MzgwRERDMTc2MT
#FDREQ2REM5ODlDRTBDOUYzMiIsICJ5YiI6ICI4NDhCMDc3OUZGNDE1RjBBRjRFQTE0
#REY5REQxRDNDMjlBQzQxRDgzNkM3ODA4ODk2QzRFQkExOUM1MUFDNDBBIiwgIkFEYi
#I6ICI0MTQ0NjIiLCAiWWIiOiAiMjQ4Q0NDRjZENUNEQzM2NDZGMEFENTkzRjlFNkNF
#RjRFNjlENDk0NUY4MzcyRTYyMzUxMkVDRUEzMjE4NTYyMyIsICJLIjogIjVCMDY3RU
#ZGQkRDMEIyQTBFMUQ5MDdCMjFFQkIyNUNGRURCOTZBODUyMTc5QTg0N0MzN0U0M0VF
#NzEzMjJDNkIiLCAiSVNLX0lSIjogIjZFMTlCODc1RjdBNTYxRDZCM0NBM0RCQjlFRj
#QyQUM1NURFM0U3MTc4ODEwMTgyMDRCODkyMkI0RDVFNTNCQjJBQTgyQzMwMEJFQTdC
#NjVEMkI2NzFEQTcxOTIyRERGNjQ3MjMwMUI3OUJDMjcwQURGQThCRjQxMzI4NUYyMj
#YzIiwgIklTS19TWSI6ICJFRUY3NDVFMkY2RTdBRTJCMUExRTUzREEzNDBFNzc3MTY3
#QTA3RkUxNTA0MzY2NDhDNTFGQjE5OUMxMUYzQ0JBQkZDNjgzQTJCNDhFMUFGNTg4MT
#k0MERDMzk4RDM3NUM5NUU2QjRBRTk5NDhBNDVCODc3MERFMDY1NjM4MkJFNCIsICJz
#aWRfb3V0cHV0X2lyIjogIkNCQzczRjYyNTg5QkJDOTZBQjZBOTVFQzIzNjNERjYyMU
#U5M0JDM0IwQ0VBODNCQTZCOTU3MUQwMDVGQThGNUQyRDA4RjcxNjU2MjI3NzdGQTQ4
#NEMwMkE5RTZCMjBBODRFRTJEQkVCQUU4QzUzQkU3NTdEQ0ZDMEVFQkRFQjVGIiwgIn
#NpZF9vdXRwdXRfb2MiOiAiM0E1MDRFOUM3RjFGN0ZBNzMxNDg2MUUyQzQ4N0QxM0Yy
#ODU2NkYzMDQzRjBDQTc2MEQyMkM0OTExQUNBMEREOEIxRjEyQTdBRDA4NjJFQjkyRD
#A4QTc2MTIwMTQwNDEyQUU2QjgzMjJFOTlENzVDRjFEMjBEOENGREUyQjQwRkUifQ==
```

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'\x0bA_initiator\x0bB_responder'
       CI = 0b415f696e69746961746f720b425f726573706f6e646572
       sid = 5223e0cdc45d6575668d64c552004124
     Outputs
       generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
       (length: 178 bytes)
         0843506163653434380850617373776f726475000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000180b415f
         696e69746961746f720b425f726573706f6e646572105223e0cdc45d
         6575668d64c552004124
       hash generator string: (length: 56 bytes)
         e129c967efb40a175ac2bf4443dfb8c2360d7fba99922b04c19e0986
         a3ef779a6fcf6942c9497adcc90c6db1f03cc0b8680f87df1fd88c35
       decoded field element of 448 bits: (length: 56 bytes)
         e129c967efb40a175ac2bf4443dfb8c2360d7fba99922b04c19e0986
         a3ef779a6fcf6942c9497adcc90c6db1f03cc0b8680f87df1fd88c35
       generator g: (length: 56 bytes)
         9ccb9b0468ea5a8a8193d9d915620f2847565d92449e939b314e76cc
         b1f88b00e5e803e5b65643c17fdba0c2ba839b09ed5873e811ff5a98
```

B.2.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQUtFLTI1NiIsICJILnNfaW5fYnl0ZXMiOiAxMzYsICJQUlMiOiAiNT
#A2MTczNzM3NzZGNzI2NCIsICJaUEFEIGxlbmd0aCI6IDExNywgIkRTSSI6ICI0MzUw
#NjE2MzY1MzQzNDM4IiwgIkNJIjogIjBCNDE1RjY5NkU2OTc0Njk2MTc0NkY3MjBCND
#I1RjcyNjU3MzcwNkY2RTY0NjU3MiIsICJzaWQiOiAiNTIyM0UwQ0RDNDVENjU3NTY2
#OEQ2NEM1NTIwMDQxMjQiLCAiZ2VuZXJhdG9yX3N0cmluZyhHLkRTSSxQUlMsQ0ksc2
#lkLEguc19pbl9ieXRlcykiOiAiMDg0MzUwNjE2MzY1MzQzNDM4MDg1MDYxNzM3Mzc3
#NkY3MjY0NzUwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAxODBCND
#E1RjY5NkU2OTc0Njk2MTc0NkY3MjBCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjEwNTIy
#M0UwQ0RDNDVENjU3NTY2OEQ2NEM1NTIwMDQxMjQiLCAiaGFzaCBnZW5lcmF0b3Igc3
#RyaW5nIjogIkUxMjlDOTY3RUZCNDBBMTc1QUMyQkY0NDQzREZCOEMyMzYwRDdGQkE5
#OTkyMkIwNEMxOUUwOTg2QTNFRjc3OUE2RkNGNjk0MkM5NDk3QURDQzkwQzZEQjFGMD
#NDQzBCODY4MEY4N0RGMUZEODhDMzUiLCAiZGVjb2RlZCBmaWVsZCBlbGVtZW50IG9m
#IDQ0OCBiaXRzIjogIkUxMjlDOTY3RUZCNDBBMTc1QUMyQkY0NDQzREZCOEMyMzYwRD
#dGQkE5OTkyMkIwNEMxOUUwOTg2QTNFRjc3OUE2RkNGNjk0MkM5NDk3QURDQzkwQzZE
#QjFGMDNDQzBCODY4MEY4N0RGMUZEODhDMzUiLCAiZ2VuZXJhdG9yIGciOiAiOUNDQj
#lCMDQ2OEVBNUE4QTgxOTNEOUQ5MTU2MjBGMjg0NzU2NUQ5MjQ0OUU5MzlCMzE0RTc2
#Q0NCMUY4OEIwMEU1RTgwM0U1QjY1NjQzQzE3RkRCQTBDMkJBODM5QjA5RUQ1ODczRT
#gxMUZGNUE5OCJ9
```

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)
    d8b2cc0fab9298a7a5c6219fb9c2e77dbdb22625f48aef974003ad9d
    3d5b58ca4e73a6b7033a3d083cd747b55c6f7345467d0a4cf82eebe0
```

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)
    89aaf7f3f3b708cecf0e1d2c0fab82817ac6c741291106414a25a474
    187abca5214ca8a3f0b55f6809cbd5316e139b2721afa70dcb0971d3
```

B.2.4. Test vector for secret points K

```
scalar_mult_vfy(ya,Yb): (length: 56 bytes)
  0fa58f732254a0708032d764d19b734b5cc890df45ec244d6d522767
  af6bdc802868688b543f1a93990a5b05dd536c49d57290f2f0bc028e
scalar_mult_vfy(yb,Ya): (length: 56 bytes)
  0fa58f732254a0708032d764d19b734b5cc890df45ec244d6d522767
  af6bdc802868688b543f1a93990a5b05dd536c49d57290f2f0bc028e
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 122 bytes)
  38d8b2cc0fab9298a7a5c6219fb9c2e77dbdb22625f48aef974003ad
  9d3d5b58ca4e73a6b7033a3d083cd747b55c6f7345467d0a4cf82eeb
  e0034144613889aaf7f3f3b708cecf0e1d2c0fab82817ac6c7412911
  06414a25a474187abca5214ca8a3f0b55f6809cbd5316e139b2721af
  a70dcb0971d303414462
DSI = G.DSI_ISK, b'CPace448_ISK': (length: 12 bytes)
  435061636534343 85f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 209 bytes)
  0c435061636534343 85f49534b105223e0cdc45d6575668d64c55200
  4124380fa58f732254a0708032d764d19b734b5cc890df45ec244d6d
  522767af6bdc802868688b543f1a93990a5b05dd536c49d57290f2f0
  bc028e38d8b2cc0fab9298a7a5c6219fb9c2e77dbdb22625f48aef97
  4003ad9d3d5b58ca4e73a6b7033a3d083cd747b55c6f7345467d0a4c
  f82eebe0034144613889aaf7f3f3b708cecf0e1d2c0fab82817ac6c7
  41291106414a25a474187abca5214ca8a3f0b55f6809cbd5316e139b
  2721afa70dcb0971d303414462
ISK result: (length: 64 bytes)
  ff3f4fa2f91e0badf5780ed2b795b97e48bb36d34c3e48cdfa1e7f79
  39ad0d6987a7fda8857b3cca3c311eaab94ce05e6d9bd111bb417ba0
  c91baf14b967eafb
```

B.2.6. Test vector for ISK calculation parallel execution

```
       transcript_oc(Ya,ADa,Yb,ADb): (length: 124 bytes)
         6f6338d8b2cc0fab9298a7a5c6219fb9c2e77dbdb22625f48aef9740
         03ad9d3d5b58ca4e73a6b7033a3d083cd747b55c6f7345467d0a4cf8
         2eebe0034144613889aaf7f3f3b708cecf0e1d2c0fab82817ac6c741
         291106414a25a474187abca5214ca8a3f0b55f6809cbd5316e139b27
         21afa70dcb0971d303414462
       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
         4124380fa58f732254a0708032d764d19b734b5cc890df45ec244d6d
         522767af6bdc802868688b543f1a93990a5b05dd536c49d57290f2f0
         bc028e6f6338d8b2cc0fab9298a7a5c6219fb9c2e77dbdb22625f48a
         ef974003ad9d3d5b58ca4e73a6b7033a3d083cd747b55c6f7345467d
         0a4cf82eebe0034144613889aaf7f3f3b708cecf0e1d2c0fab82817a
         c6c741291106414a25a474187abca5214ca8a3f0b55f6809cbd5316e
         139b2721afa70dcb0971d303414462
       ISK result: (length: 64 bytes)
         0ab598d085cd0bcdca64b58630e53287ef5aa7dbfc209dc9d2e50e0c
         6eb8e5e5ac27987d48e5608a45228c24532821d00fe66c7acc72fccf
         2d14d95d4ee645ed
```

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

```
       H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         6ef5882a6a9a323789e418a27c254513412e3c75945e1c0777dc2725
         0a122bc1c64106aedc800a0cd51d35203d2ae09471c1a81448b70a2e
         3715376d9f03b79b
       H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         5784ff2c7c3dce998db4976cbeccfe0ec3a253a2caf64ae4132f5394
         0390ab5938dbd3c0fddc6d070f8fde384d49898873146b2921ee92a6
         74ac1ca45ca365f4
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
    0x9c,0xcb,0x9b,0x04,0x68,0xea,0x5a,0x8a,0x81,0x93,0xd9,0xd9,
    0x15,0x62,0x0f,0x28,0x47,0x56,0x5d,0x92,0x44,0x9e,0x93,0x9b,
    0x31,0x4e,0x76,0xcc,0xb1,0xf8,0x8b,0x00,0xe5,0xe8,0x03,0xe5,
    0xb6,0x56,0x43,0xc1,0x7f,0xdb,0xa0,0xc2,0xba,0x83,0x9b,0x09,
    0xed,0x58,0x73,0xe8,0x11,0xff,0x5a,0x98,
   };
   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[] = {
    0xd8,0xb2,0xcc,0x0f,0xab,0x92,0x98,0xa7,0xa5,0xc6,0x21,0x9f,
    0xb9,0xc2,0xe7,0x7d,0xbd,0xb2,0x26,0x25,0xf4,0x8a,0xef,0x97,
    0x40,0x03,0xad,0x9d,0x3d,0x5b,0x58,0xca,0x4e,0x73,0xa6,0xb7,
    0x03,0x3a,0x3d,0x08,0x3c,0xd7,0x47,0xb5,0x5c,0x6f,0x73,0x45,
    0x46,0x7d,0x0a,0x4c,0xf8,0x2e,0xeb,0xe0,
   };
   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[] = {
    0x89,0xaa,0xf7,0xf3,0xf3,0xb7,0x08,0xce,0xcf,0x0e,0x1d,0x2c,
    0x0f,0xab,0x82,0x81,0x7a,0xc6,0xc7,0x41,0x29,0x11,0x06,0x41,
    0x4a,0x25,0xa4,0x74,0x18,0x7a,0xbc,0xa5,0x21,0x4c,0xa8,0xa3,
    0xf0,0xb5,0x5f,0x68,0x09,0xcb,0xd5,0x31,0x6e,0x13,0x9b,0x27,
    0x21,0xaf,0xa7,0x0d,0xcb,0x09,0x71,0xd3,
   };
   const unsigned char tc_K[] = {
    0x0f,0xa5,0x8f,0x73,0x22,0x54,0xa0,0x70,0x80,0x32,0xd7,0x64,
    0xd1,0x9b,0x73,0x4b,0x5c,0xc8,0x90,0xdf,0x45,0xec,0x24,0x4d,
    0x6d,0x52,0x27,0x67,0xaf,0x6b,0xdc,0x80,0x28,0x68,0x68,0x8b,
    0x54,0x3f,0x1a,0x93,0x99,0x0a,0x5b,0x05,0xdd,0x53,0x6c,0x49,
    0xd5,0x72,0x90,0xf2,0xf0,0xbc,0x02,0x8e,
```

```
};
const unsigned char tc_ISK_IR[] = {
 0xff,0x3f,0x4f,0xa2,0xf9,0x1e,0x0b,0xad,0xf5,0x78,0x0e,0xd2,
 0xb7,0x95,0xb9,0x7e,0x48,0xbb,0x36,0xd3,0x4c,0x3e,0x48,0xcd,
 0xfa,0x1e,0x7f,0x79,0x39,0xad,0x0d,0x69,0x87,0xa7,0xfd,0xa8,
 0x85,0x7b,0x3c,0xca,0x3c,0x31,0x1e,0xaa,0xb9,0x4c,0xe0,0x5e,
 0x6d,0x9b,0xd1,0x11,0xbb,0x41,0x7b,0xa0,0xc9,0x1b,0xaf,0x14,
 0xb9,0x67,0xea,0xfb,
};
const unsigned char tc_ISK_SY[] = {
 0x0a,0xb5,0x98,0xd0,0x85,0xcd,0x0b,0xcd,0xca,0x64,0xb5,0x86,
 0x30,0xe5,0x32,0x87,0xef,0x5a,0xa7,0xdb,0xfc,0x20,0x9d,0xc9,
 0xd2,0xe5,0x0e,0x0c,0x6e,0xb8,0xe5,0xe5,0xac,0x27,0x98,0x7d,
 0x48,0xe5,0x60,0x8a,0x45,0x22,0x8c,0x24,0x53,0x28,0x21,0xd0,
 0x0f,0xe6,0x6c,0x7a,0xcc,0x72,0xfc,0xcf,0x2d,0x14,0xd9,0x5d,
 0x4e,0xe6,0x45,0xed,
};
const unsigned char tc_sid_out_ir[] = {
 0x6e,0xf5,0x88,0x2a,0x6a,0x9a,0x32,0x37,0x89,0xe4,0x18,0xa2,
 0x7c,0x25,0x45,0x13,0x41,0x2e,0x3c,0x75,0x94,0x5e,0x1c,0x07,
 0x77,0xdc,0x27,0x25,0x0a,0x12,0x2b,0xc1,0xc6,0x41,0x06,0xae,
 0xdc,0x80,0x0a,0x0c,0xd5,0x1d,0x35,0x20,0x3d,0x2a,0xe0,0x94,
 0x71,0xc1,0xa8,0x14,0x48,0xb7,0x0a,0x2e,0x37,0x15,0x37,0x6d,
 0x9f,0x03,0xb7,0x9b,
};
const unsigned char tc_sid_out_oc[] = {
 0x57,0x84,0xff,0x2c,0x7c,0x3d,0xce,0x99,0x8d,0xb4,0x97,0x6c,
 0xbe,0xcc,0xfe,0x0e,0xc3,0xa2,0x53,0xa2,0xca,0xf6,0x4a,0xe4,
 0x13,0x2f,0x53,0x94,0x03,0x90,0xab,0x59,0x38,0xdb,0xd3,0xc0,
 0xfd,0xdc,0x6d,0x07,0x0f,0x8f,0xde,0x38,0x4d,0x49,0x89,0x88,
 0x73,0x14,0x6b,0x29,0x21,0xee,0x92,0xa6,0x74,0xac,0x1c,0xa4,
 0x5c,0xa3,0x65,0xf4,
};
```

B.2.9. Testvectors as JSON file encoded as BASE64

```
#eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
#Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjUyMjNF
#MENEQzQ1RDY1NzU2NjhENjRDNTUyMDA0MTI0IiwgImciOiAiOUNDQjlCMDQ2OEVBNU
#E4QTgxOTNEOUQ5MTU2MjBGMjg0NzU2NUQ5MjQ0OUU5MzlCMzE0RTc2Q0NCMUY4OEIw
#MEU1RTgwM0U1QjY1NjQzQzE3RkRCQTBDMkJBODM5QjA5RUQ1ODczRTgxMUZGNUE5OC
#IsICJ5YSI6ICIyMUI0RjRCRDlFNjRFRDM1NUMzRUI2NzZBMjhFQkVEQUY2RDhGMTdC
#REMzNjU5OTVCMzE5MDk3MTUzMDQ0MDgwNTE2QkQwODNCRkNDRTY2MTIxQTMwNzI2ND
#Y5OTRDODQzMENDMzgyQjhEQzU0M0U4IiwgIkFEYSI6ICI0MTQ0NjEiLCAiWWEiOiAi
#RDhCMkNDMEZBQjkyOThBN0E1QzYyMTlGQjlDMkU3N0RCREIyMjYyNUY0OEFFRjk3ND
#AwM0FEOUQzRDVCNThDQTRFNzNBNkI3MDMzQTNEMDgzQ0Q3NDdCNTVDNkY3MzQ1NDY3
#RDBBNENGODJFRUJFMCIsICJ5YiI6ICI4NDhCMDc3OUZGNDE1RjBBRjRFQTE0REY5RE
#QxRDNDMjlBQzQxRDgzNkM3ODA4ODk2QzRFQkExOUM1MUFDNDBBNDM5Q0FGNUU2MUVD
#ODhDMzA3QzdENjE5MTk1MjI5NDEyRUFBNzNGQjJBNUVBMjBEIiwgIkFEYiI6ICI0MT
#Q0NjIiLCAiWWIiOiAiODlBQUY3RjNGM0I3MDhDRUNGMEUxRDJDMEZBQjgyODE3QUM2
#Qzc0MTI5MTEwNjQxNEEyNUE0NzQxODdBQkNBNTIxNENBOEEzRjBCNTVGNjgwOUNCRD
#UzMTZFMTM5QjI3MjFBRkE3MERDQjA5NzFEMyIsICJLIjogIjBGQTU4RjczMjI1NEEw
#NzA4MDMyRDc2NEQxOUI3MzRCNUNDODkwREY0NUVDMjQ0RDZENTIyNzY3QUY2QkRDOD
#AyODY4Njg4QjU0M0YxQTkzOTkwQTVCMDVERDUzNkM0OUQ1NzI5MEYyRjBCQzAyOEUi
#LCAiSVNLX0lSIjogIkZGM0Y0RkEyRjkxRTBCQURGNTc4MEVEMkI3OTVCOTdFNDhCQj
#M2RDM0QzNFNDhDREZBMUU3Rjc5MzlBRDBENjk4N0E3RkRBODg1N0IzQ0NBM0MzMTFF
#QUFCOTRDRTA1RTZEOUJEMTExQkI0MTdCQTBDOTFCQUYxNEI5NjdFQUZCIiwgIklTS1
#9TWSI6ICIwQUI1OThEMDg1Q0QwQkNEQ0E2NEI1ODYzMEU1MzI4N0VGNUFBN0RCRkMy
#MDlEQzlEMkU1MEUwQzZFQjhFNUU1QUMyNzk4N0Q0OEU1NjA4QTQ1MjI4QzI0NTMyOD
#IxRDAwRkU2NkM3QUNDNzJGQ0NGMkQxNEQ5NUQ0RUU2NDVFRCIsICJzaWRfb3V0cHV0
#X2lyIjogIjZFRjU4ODJBNkE5QTMyMzc4OUU0MThBMjdDMjU0NTEzNDEyRTNDNzU5ND
#VFMUMwNzc3REMyNzI1MEExMjJCQzFDNjQxMDZBRURDODAwQTBDRDUxRDM1MjAzRDJB
#RTA5NDcxQzFBODE0NDhCNzBBMkUzNzE1Mzc2RDlGMDNCNzlCIiwgInNpZF9vdXRwdX
#Rfb2MiOiAiNTc4NEZGMkM3QzNEQ0U5OThEQjQ5NzZDQkVDQ0ZFMEVDM0EyNTNBMkNB
#RjY0QUU0MTMyRjUzOTQwMzkwQUI1OTM4REJEM0MwRkREQzZEMDcwRjhGREUzODREND
#k4OTg4NzMxNDZCMjkyMUVFOTJBNjc0QUMxQ0E0NUNBMzY1RjQifQ==
```

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'\x0bA_initiator\x0bB_responder'
  CI = 0b415f696e69746961746f720b425f726573706f6e646572
  sid = 7e4b4791d6a8ef019b936c79fb7f2c57
Outputs
  generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
  (length: 170 bytes)
    1143506163655269737472657474 6f32353085061737377 6f726464
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000
    000000000000000000000000000180b415f696e69746961746f
    720b425f726573706f6e646572107e4b4791d6a8ef019b936c79fb7f
    2c57
  hash result: (length: 64 bytes)
    da6d3ddc8802fca9058755ffd3ebde08a9c2c74945901a258482a288
    b6663af06bf645c93cd1c51512307199c80e84908916d983b34af772
    05f90851a657ee27
  encoded generator g: (length: 32 bytes)
    222b6b195fe84b1652badb6f6a3ae3d24341e7306967f0b8115b40d5
    698c7e56
```

B.3.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS01MTIiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
#E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiAxMDAsICJEU0kiOiAiNDM1MDYx
#NjM2NTUyNjk3Mzc0NzI2NTc0NzQ2RjMyMzUzNSIsICJDSSI6ICIwQjQxNUY2OTZFNj
#k3NDY5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjdF
#NEI0NzkxRDZBOEVGMDE5QjkzNkM3OUZCN0YyQzU3IiwgImdlbmVyYXRvcl9zdHJpbm
#coRy5EU0ksUFJTLENJLHNpZCxILnNfaW5fYnl0ZXMpIjogIjExNDM1MDYxNjM2NTUy
#Njk3Mzc0NzI2NTc0NzQ2RjMyMzUzNTA4NTA2MTczNzM3NzZGNzI2NDY0MDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAxODBCND
#E1RjY5NkU2OTc0Njk2MTc0NkY3MjBCNDI1RjcyNjU3MzcwNkY2RTY0NjU3MjEwN0U0
#QjQ3OTFENkE4RUYwMTlCOTM2Qzc5RkI3RjJDNTciLCAiaGFzaCByZXN1bHQiOiAiRE
#E2RDNEREM4ODAyRkNBOTA1ODc1NUZGRDNFQkRFMDhBOUMyQzc0OTQ1OTAxQTI1ODQ4
#MkEyODhCNjY2M0FGMDZCRjY0NUM5M0NEMUM1MTUxMjMwNzE5OUM4MEU4NDkwODkxNk
#Q5ODNCMzRBRjc3MjA1RjkwODUxQTY1N0VFMjciLCAiZW5jb2RlZCBnZW5lcmF0b3Ig
#ZyI6ICIyMjJCNkIxOTVGRTg0QjE2NTJCQURCNkY2QTNBRTNEMjQzNDFFNzMwNjk2N0
#YwQjgxMTVCNDBENTY5OEM3RTU2In0=
```

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)
         d6bac480f2c386c394efc7c47adb9925dcd2630b64f240c50f8d0eec
         482b9157
```

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)
         3ea7e0b19560d7c0b0f5734f63b955286dfa8232b5ebe63324e2d9e7
         433f7258
```

B.3.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 32 bytes)
         80b69a8a76457ab6a4d7f887a4bf6b55a2f80ac19c333f917a05fc98
         87c8b40f
       scalar_mult_vfy(yb,Ya): (length: 32 bytes)
         80b69a8a76457ab6a4d7f887a4bf6b55a2f80ac19c333f917a05fc98
         87c8b40f
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 74 bytes)
  20d6bac480f2c386c394efc7c47adb9925dcd2630b64f240c50f8d0e
  ec482b915703414461203ea7e0b19560d7c0b0f5734f63b955286dfa
  8232b5ebe63324e2d9e7433f725803414462
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
  a8ef019b936c79fb7f2c572080b69a8a76457ab6a4d7f887a4bf6b55
  a2f80ac19c333f917a05fc9887c8b40f20d6bac480f2c386c394efc7
  c47adb9925dcd2630b64f240c50f8d0eec482b915703414461203ea7
  e0b19560d7c0b0f5734f63b955286dfa8232b5ebe63324e2d9e7433f
  725803414462
ISK result: (length: 64 bytes)
  b69effbf61b51d56401c0f65601abe428de8206feaaf0e32198896dc
  ae7b35cd2b38950a39dfd5d4a79164614c2984f7daa460b588c1e80c
  3fa2068af7900447
```

B.3.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 76 bytes)
  6f6320d6bac480f2c386c394efc7c47adb9925dcd2630b64f240c50f
  8d0eec482b915703414461203ea7e0b19560d7c0b0f5734f63b95528
  6dfa8232b5ebe63324e2d9e7433f725803414462
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
  a8ef019b936c79fb7f2c572080b69a8a76457ab6a4d7f887a4bf6b55
  a2f80ac19c333f917a05fc9887c8b40f6f6320d6bac480f2c386c394
  efc7c47adb9925dcd2630b64f240c50f8d0eec482b91570341446120
  3ea7e0b19560d7c0b0f5734f63b955286dfa8232b5ebe63324e2d9e7
  433f725803414462
ISK result: (length: 64 bytes)
  544199d71f62f8d9a1fee55727e24fe4a45844593c2b6013c4fa3969
  d0e5debb2244675c0b43397cbb68d342b01fc0f98fc961469a25134d
  e9f0f813c1a57476
```

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

```
      H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        bb1c449b35f0ea79a65c209f329a693d475e0ce2387bed9fe4b78f60
        b2a27c219813fb2cfe175ef40d2222d9261e66da7d78f7c55a303b1b
        8611dcdfab880c47
      H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        10d5941d4933497fe31b9188d690b84465e2a2d158332a7267284a07
        1a8d0876fc5c8c329dc735d59a9f8ef6623ee23924704a2f929dd631
        ca981227ee82fff2
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
    0x22,0x2b,0x6b,0x19,0x5f,0xe8,0x4b,0x16,0x52,0xba,0xdb,0x6f,
    0x6a,0x3a,0xe3,0xd2,0x43,0x41,0xe7,0x30,0x69,0x67,0xf0,0xb8,
    0x11,0x5b,0x40,0xd5,0x69,0x8c,0x7e,0x56,
   };
   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[] = {
    0xd6,0xba,0xc4,0x80,0xf2,0xc3,0x86,0xc3,0x94,0xef,0xc7,0xc4,
    0x7a,0xdb,0x99,0x25,0xdc,0xd2,0x63,0x0b,0x64,0xf2,0x40,0xc5,
    0x0f,0x8d,0x0e,0xec,0x48,0x2b,0x91,0x57,
   };
   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[] = {
    0x3e,0xa7,0xe0,0xb1,0x95,0x60,0xd7,0xc0,0xb0,0xf5,0x73,0x4f,
    0x63,0xb9,0x55,0x28,0x6d,0xfa,0x82,0x32,0xb5,0xeb,0xe6,0x33,
    0x24,0xe2,0xd9,0xe7,0x43,0x3f,0x72,0x58,
   };
   const unsigned char tc_K[] = {
    0x80,0xb6,0x9a,0x8a,0x76,0x45,0x7a,0xb6,0xa4,0xd7,0xf8,0x87,
    0xa4,0xbf,0x6b,0x55,0xa2,0xf8,0x0a,0xc1,0x9c,0x33,0x3f,0x91,
    0x7a,0x05,0xfc,0x98,0x87,0xc8,0xb4,0x0f,
   };
   const unsigned char tc_ISK_IR[] = {
    0xb6,0x9e,0xff,0xbf,0x61,0xb5,0x1d,0x56,0x40,0x1c,0x0f,0x65,
    0x60,0x1a,0xbe,0x42,0x8d,0xe8,0x20,0x6f,0xea,0xaf,0x0e,0x32,
    0x19,0x88,0x96,0xdc,0xae,0x7b,0x35,0xcd,0x2b,0x38,0x95,0x0a,
    0x39,0xdf,0xd5,0xd4,0xa7,0x91,0x64,0x61,0x4c,0x29,0x84,0xf7,
    0xda,0xa4,0x60,0xb5,0x88,0xc1,0xe8,0x0c,0x3f,0xa2,0x06,0x8a,
    0xf7,0x90,0x04,0x47,
   };
   const unsigned char tc_ISK_SY[] = {
    0x54,0x41,0x99,0xd7,0x1f,0x62,0xf8,0xd9,0xa1,0xfe,0xe5,0x57,
    0x27,0xe2,0x4f,0xe4,0xa4,0x58,0x44,0x59,0x3c,0x2b,0x60,0x13,
    0xc4,0xfa,0x39,0x69,0xd0,0xe5,0xde,0xbb,0x22,0x44,0x67,0x5c,
    0x0b,0x43,0x39,0x7c,0xbb,0x68,0xd3,0x42,0xb0,0x1f,0xc0,0xf9,
    0x8f,0xc9,0x61,0x46,0x9a,0x25,0x13,0x4d,0xe9,0xf0,0xf8,0x13,
    0xc1,0xa5,0x74,0x76,
   };
   const unsigned char tc_sid_out_ir[] = {
    0xbb,0x1c,0x44,0x9b,0x35,0xf0,0xea,0x79,0xa6,0x5c,0x20,0x9f,
    0x32,0x9a,0x69,0x3d,0x47,0x5e,0x0c,0xe2,0x38,0x7b,0xed,0x9f,
    0xe4,0xb7,0x8f,0x60,0xb2,0xa2,0x7c,0x21,0x98,0x13,0xfb,0x2c,
    0xfe,0x17,0x5e,0xf4,0x0d,0x22,0x22,0xd9,0x26,0x1e,0x66,0xda,
    0x7d,0x78,0xf7,0xc5,0x5a,0x30,0x3b,0x1b,0x86,0x11,0xdc,0xdf,
    0xab,0x88,0x0c,0x47,
   };
   const unsigned char tc_sid_out_oc[] = {
    0x10,0xd5,0x94,0x1d,0x49,0x33,0x49,0x7f,0xe3,0x1b,0x91,0x88,
    0xd6,0x90,0xb8,0x44,0x65,0xe2,0xa2,0xd1,0x58,0x33,0x2a,0x72,
    0x67,0x28,0x4a,0x07,0x1a,0x8d,0x08,0x76,0xfc,0x5c,0x8c,0x32,
    0x9d,0xc7,0x35,0xd5,0x9a,0x9f,0x8e,0xf6,0x62,0x3e,0xe2,0x39,
    0x24,0x70,0x4a,0x2f,0x92,0x9d,0xd6,0x31,0xca,0x98,0x12,0x27,
    0xee,0x82,0xff,0xf2,
   };
```

B.3.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
    #Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjdFNEI0
    #NzkxRDZBOEVGMDE5QjkzNkM3OUZCN0YyQzU3IiwgImciOiAiMjIyQjZCMTk1RkU4NE
    #IxNjUyQkFEQjZGNkEzQUUzRDI0MzQxRTczMDY5NjdGMEI4MTE1QjQwRDU2OThDN0U1
    #NiIsICJ5YSI6ICJEQTNEMjM3MDBBOUU1Njk5MjU4QUVGOTREQzA2MERGREE1RUJCNj
    #FGMDJBNUVBNzdGQUQ1M0Y0RkYwOTc2RDA4IiwgIkFEYSI6ICI0MTQ0NjEiLCAiWWEi
    #OiAiRDZCQUM0ODBGMkMzODZDMzk0RUZDN0M0N0FEQjk5MjVEQ0QyNjMwQjY0RjI0ME
    #M1MEY4RDBFRUM0ODJCOTE1NyIsICJ5YiI6ICJEMjMxNkI0NTQ3MThDMzUzNjJEODNE
    #NjlERjYzMjBGMzg1NzhFRDU5ODQ2NTE0MzVFMjk0OTc2MkQ5MDBCODBEIiwgIkFEYi
    #I6ICI0MTQ0NjIiLCAiWWIiOiAiM0VBN0UwQjE5NTYwRDdDMEIwRjU3MzRGNjNCOTU1
    #Mjg2REZBODIzMkI1RUJFNjMzMjRFMkQ5RTc0MzNGNzI1OCIsICJLIjogIjgwQjY5QT
    #hBNzY0NTdBQjZBNEQ3Rjg4N0E0QkY2QjU1QTJGODBBQzE5QzMzM0Y5MTdBMDVGQzk4
    #ODdDOEI0MEYiLCAiSVNLX0lSIjogIkI2OUVGRkJGNjFCNTFENTY0MDFDMEY2NTYwMU
    #FCRTQyOERFODIwNkZFQUFGMEUzMjE5ODg5NkRDQUU3QjM1Q0QyQjM4OTUwQTM5REZE
    #NUQ0QTc5MTY0NjE0QzI5ODRGN0RBQTQ2MEI1ODhDMUU4MEMzRkEyMDY4QUY3OTAwND
    #Q3IiwgIklTS19TWSI6ICI1NDQxOTlENzFGNjJGOEQ5QTFGRUU1NTcyN0UyNEZFNEE0
    #NTg0NDU5M0MyQjYwMTNDNEZBMzk2OUQwRTVERUJCMjI0NDY3NUMwQjQzMzk3Q0JCNj
    #hEMzQyQjAxRkMwRjk4RkM5NjE0NjlBMjUxMzRERTlGMEY4MTNDMUE1NzQ3NiIsICJz
    #aWRfb3V0cHV0X2lyIjogIkJCMUM0NDlCMzVGMEVBNzlBNjVDMjA5RjMyOUE2OTNEND
    #c1RTBDRTIzODdCRUQ5RkU0Qjc4RjYwQjJBMjdDMjE5ODEzRkIyQ0ZFMTc1RUY0MEQy
    #MjIyRDkyNjFFNjZEQTdENzhGN0M1NUEzMDNCMUI4NjExRENERkFCODgwQzQ3IiwgIn
    #NpZF9vdXRwdXRfb2MiOiAiMTBENTk0MUQ0OTMzNDk3RkUzMUI5MTg4RDY5MEI4NDQ2
    #NUUyQTJEMTU4MzMyQTcyNjcyODRBMDcxQThEMDg3NkZDNUM4QzMyOURDNzM1RDU5QT
    #lGOEVGNjYyM0VFMjM5MjQ3MDRBMkY5MjlERDYzMUNBOTgxMjI3RUU4MkZGRjIifQ==
```

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)
        0000000000000000000000000000000000000000000000000000000
        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'\x0bA_initiator\x0bB_responder'
       CI = 0b415f696e69746961746f720b425f726573706f6e646572
       sid = 5223e0cdc45d6575668d64c552004124
     Outputs
       generator_string(G.DSI,PRS,CI,sid,H.s_in_bytes):
       (length: 178 bytes)
         0d435061636544656361663434380850617373776f72647000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000180b415f
         696e69746961746f720b425f726573706f6e646572105223e0cdc45d
         6575668d64c552004124
       hash result: (length: 112 bytes)
         8ad1e4999548af985785a92cf36c5ba8b75115c7b38359ddb45a7347
         ab381673a3aeaac0edf6aaa666f9f4362d0115d8b9ead3e3b0235de1
         4032c6952ac33ea84af04081bda042548c7bde48a2950cd5a370b786
         e2e5ef832b113f0ba86f46954ea5f1f99b50ce7e65d35ca3a8f93bf6
       encoded generator g: (length: 56 bytes)
         b47b2c3e328ea9f318ca695af65dc838ba5a58bdb8087ae6c6ab486a
         8bfcb13d35b301dddca91d1540a4106d38ba1bf152e5bba104193c7e
```

B.4.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQUtFLTI1NiIsICJILnNfaW5fYnl0ZXMiOiAxMzYsICJQUlMiOiAiNT
#A2MTczNzM3NzZGNzI2NCIsICJaUEFEIGxlbmd0aCI6IDExMiwgIkRTSSI6ICI0MzUw
#NjE2MzY1NDQ2NTYzNjE2NjM0MzQzOCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3NDY5Nj
#E3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjUyMjNFMENE
#QzQ1RDY1NzU2NjhENjRDNTUyMDA0MTI0IiwgImdlbmVyYXRvcl9zdHJpbmcoRy5EU0
#ksUFJTLENJLHNpZCxILnNfaW5fYnl0ZXMpIjogIjBENDM1MDYxNjM2NTQ0NjU2MzYx
#NjYzNDM0MzgwODUwNjE3MzczNzc2RjcyNjQ3MDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMTgwQjQxNUY2OTZFNjk3NDY5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2
#NDY1NzIxMDUyMjNFMENEQzQ1RDY1NzU2NjhENjRDNTUyMDA0MTI0IiwgImhhc2ggcm
#VzdWx0IjogIjhBRDFFNDk5OTU0OEFGOTg1Nzg1QTkyQ0YzNkM1QkE4Qjc1MTE1QzdC
#MzgzNTlEREI0NUE3MzQ3QUIzODE2NzNBM0FFQUFDMEVERjZBQUE2NjZGOUY0MzYyRD
#AxMTVEOEI5RUFEM0UzQjAyMzVERTE0MDMyQzY5NTJBQzMzRUE4NEFGMDQwODFCREEw
#NDI1NDhDN0JERTQ4QTI5NTBDRDVBMzcwQjc4NkUyRTVFRjgzMkIxMTNGMEJBODZGND
#Y5NTRFQTVGMUY5OUI1MENFN0U2NUQzNUNBM0E4RjkzQkY2IiwgImVuY29kZWQgZ2Vu
#ZXJhdG9yIGciOiAiQjQ3QjJDM0UzMjhFQTlGMzE4Q0E2OTVBRjY1REM4MzhCQTVBNT
#hCREI4MDg3QUU2QzZBQjQ4NkE4QkZDQjEzRDM1QjMwMURERENBOTFEMTU0MEE0MTA2
#RDM4QkExQkYxNTJFNUJCQTEwNDE5M0M3RSJ9
```

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)
    d2926adf737d8dfe46ae47cdd0a56bbcdad96958512d39f9865ad1b6
    4b5304d4d1696983cd6d38d57e3cdb1080262198b5d4c497c64926a5
```

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)
    2e620e5d0bc4637f1ff97ddccb015141b2c3241b4abcb5931a7e22a9
    9fe256ae543733ce4e4c41826dda142fbfeb2639727a5233f8f0a42a
```

B.4.4. Test vector for secret points K

```
scalar_mult_vfy(ya,Yb): (length: 56 bytes)
  ee0c0ead075ed5637e167cd3e16415c114aee6e21adaddc0de9151b2
  930833e2862f4e4ec30c91de5c5249a724d9114860f2bcbbb6ca83fd
scalar_mult_vfy(yb,Ya): (length: 56 bytes)
  ee0c0ead075ed5637e167cd3e16415c114aee6e21adaddc0de9151b2
  930833e2862f4e4ec30c91de5c5249a724d9114860f2bcbbb6ca83fd
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 122 bytes)
  38d2926adf737d8dfe46ae47cdd0a56bbcdad96958512d39f9865ad1
  b64b5304d4d1696983cd6d38d57e3cdb1080262198b5d4c497c64926
  a503414461382e620e5d0bc4637f1ff97ddccb015141b2c3241b4abc
  b5931a7e22a99fe256ae543733ce4e4c41826dda142fbfeb2639727a
  5233f8f0a42a03414462
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
  8d64c55200412438ee0c0ead075ed5637e167cd3e16415c114aee6e2
  1adaddc0de9151b2930833e2862f4e4ec30c91de5c5249a724d91148
  60f2bcbbb6ca83fd38d2926adf737d8dfe46ae47cdd0a56bbcdad969
  58512d39f9865ad1b64b5304d4d1696983cd6d38d57e3cdb10802621
  98b5d4c497c64926a503414461382e620e5d0bc4637f1ff97ddccb01
  5141b2c3241b4abcb5931a7e22a99fe256ae543733ce4e4c41826dda
  142fbfeb2639727a5233f8f0a42a03414462
ISK result: (length: 64 bytes)
  6f3a66ce20e1bad279d32d65844cd66b7352eb61a9eb37631168fe80
  3c83c6601c6a3e0867f4b2dbe2b0cb1b530912892d2b5eb9af2e3ab3
  d8682a33e178d1d2
```

B.4.6. Test vector for ISK calculation parallel execution

```
       transcript_oc(Ya,ADa,Yb,ADb): (length: 124 bytes)
         6f6338d2926adf737d8dfe46ae47cdd0a56bbcdad96958512d39f986
         5ad1b64b5304d4d1696983cd6d38d57e3cdb1080262198b5d4c497c6
         4926a503414461382e620e5d0bc4637f1ff97ddccb015141b2c3241b
         4abcb5931a7e22a99fe256ae543733ce4e4c41826dda142fbfeb2639
         727a5233f8f0a42a03414462
       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
         8d64c55200412438ee0c0ead075ed5637e167cd3e16415c114aee6e2
         1adaddc0de9151b2930833e2862f4e4ec30c91de5c5249a724d91148
         60f2bcbbb6ca83fd6f6338d2926adf737d8dfe46ae47cdd0a56bbcda
         d96958512d39f9865ad1b64b5304d4d1696983cd6d38d57e3cdb1080
         262198b5d4c497c64926a503414461382e620e5d0bc4637f1ff97ddc
         cb015141b2c3241b4abcb5931a7e22a99fe256ae543733ce4e4c4182
         6dda142fbfeb2639727a5233f8f0a42a03414462
       ISK result: (length: 64 bytes)
         59012b30b7a4050d1e83f7157da6cdd5baf77bf432f545e7baa1a2a1
         0bc29ef1345f9cfe2fb448bfe8a48a470d90d94cd73e4c99fc1c8b65
         1d43fb3ebed3a88f
```

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

```
       H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         9b3fb936e2b9d00da70841392bf012f3d0c1bcf0a8861020f7882b0d
         7f1e54eaf6e4d850ea4ed3169e630f2968d69e19338ae23d74febf00
         3b41600b0de91ad6
       H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
       (length: 64 bytes)
         dfdf19db27b99dbf0a8943ee6a5395b48664e3930bdb0cf5a4e79646
         52e2ceafaee3645ee668a047f3fb259affac4eaec76014c2a01d2064
         9e96a19abde0cd22
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
 0xb4,0x7b,0x2c,0x3e,0x32,0x8e,0xa9,0xf3,0x18,0xca,0x69,0x5a,
 0xf6,0x5d,0xc8,0x38,0xba,0x5a,0x58,0xbd,0xb8,0x08,0x7a,0xe6,
 0xc6,0xab,0x48,0x6a,0x8b,0xfc,0xb1,0x3d,0x35,0xb3,0x01,0xdd,
 0xdc,0xa9,0x1d,0x15,0x40,0xa4,0x10,0x6d,0x38,0xba,0x1b,0xf1,
 0x52,0xe5,0xbb,0xa1,0x04,0x19,0x3c,0x7e,
};
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[] = {
 0xd2,0x92,0x6a,0xdf,0x73,0x7d,0x8d,0xfe,0x46,0xae,0x47,0xcd,
 0xd0,0xa5,0x6b,0xbc,0xda,0xd9,0x69,0x58,0x51,0x2d,0x39,0xf9,
 0x86,0x5a,0xd1,0xb6,0x4b,0x53,0x04,0xd4,0xd1,0x69,0x69,0x83,
 0xcd,0x6d,0x38,0xd5,0x7e,0x3c,0xdb,0x10,0x80,0x26,0x21,0x98,
 0xb5,0xd4,0xc4,0x97,0xc6,0x49,0x26,0xa5,
};
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[] = {
 0x2e,0x62,0x0e,0x5d,0x0b,0xc4,0x63,0x7f,0x1f,0xf9,0x7d,0xdc,
 0xcb,0x01,0x51,0x41,0xb2,0xc3,0x24,0x1b,0x4a,0xbc,0xb5,0x93,
 0x1a,0x7e,0x22,0xa9,0x9f,0xe2,0x56,0xae,0x54,0x37,0x33,0xce,
 0x4e,0x4c,0x41,0x82,0x6d,0xda,0x14,0x2f,0xbf,0xeb,0x26,0x39,
 0x72,0x7a,0x52,0x33,0xf8,0xf0,0xa4,0x2a,
};
const unsigned char tc_K[] = {
 0xee,0x0c,0x0e,0xad,0x07,0x5e,0xd5,0x63,0x7e,0x16,0x7c,0xd3,
 0xe1,0x64,0x15,0xc1,0x14,0xae,0xe6,0xe2,0x1a,0xda,0xdd,0xc0,
 0xde,0x91,0x51,0xb2,0x93,0x08,0x33,0xe2,0x86,0x2f,0x4e,0x4e,
 0xc3,0x0c,0x91,0xde,0x5c,0x52,0x49,0xa7,0x24,0xd9,0x11,0x48,
 0x60,0xf2,0xbc,0xbb,0xb6,0xca,0x83,0xfd,
```

```
};
const unsigned char tc_ISK_IR[] = {
 0x6f,0x3a,0x66,0xce,0x20,0xe1,0xba,0xd2,0x79,0xd3,0x2d,0x65,
 0x84,0x4c,0xd6,0x6b,0x73,0x52,0xeb,0x61,0xa9,0xeb,0x37,0x63,
 0x11,0x68,0xfe,0x80,0x3c,0x83,0xc6,0x60,0x1c,0x6a,0x3e,0x08,
 0x67,0xf4,0xb2,0xdb,0xe2,0xb0,0xcb,0x1b,0x53,0x09,0x12,0x89,
 0x2d,0x2b,0x5e,0xb9,0xaf,0x2e,0x3a,0xb3,0xd8,0x68,0x2a,0x33,
 0xe1,0x78,0xd1,0xd2,
};
const unsigned char tc_ISK_SY[] = {
 0x59,0x01,0x2b,0x30,0xb7,0xa4,0x05,0x0d,0x1e,0x83,0xf7,0x15,
 0x7d,0xa6,0xcd,0xd5,0xba,0xf7,0x7b,0xf4,0x32,0xf5,0x45,0xe7,
 0xba,0xa1,0xa2,0xa1,0x0b,0xc2,0x9e,0xf1,0x34,0x5f,0x9c,0xfe,
 0x2f,0xb4,0x48,0xbf,0xe8,0xa4,0x8a,0x47,0x0d,0x90,0xd9,0x4c,
 0xd7,0x3e,0x4c,0x99,0xfc,0x1c,0x8b,0x65,0x1d,0x43,0xfb,0x3e,
 0xbe,0xd3,0xa8,0x8f,
};
const unsigned char tc_sid_out_ir[] = {
 0x9b,0x3f,0xb9,0x36,0xe2,0xb9,0xd0,0x0d,0xa7,0x08,0x41,0x39,
 0x2b,0xf0,0x12,0xf3,0xd0,0xc1,0xbc,0xf0,0xa8,0x86,0x10,0x20,
 0xf7,0x88,0x2b,0x0d,0x7f,0x1e,0x54,0xea,0xf6,0xe4,0xd8,0x50,
 0xea,0x4e,0xd3,0x16,0x9e,0x63,0x0f,0x29,0x68,0xd6,0x9e,0x19,
 0x33,0x8a,0xe2,0x3d,0x74,0xfe,0xbf,0x00,0x3b,0x41,0x60,0x0b,
 0x0d,0xe9,0x1a,0xd6,
};
const unsigned char tc_sid_out_oc[] = {
 0xdf,0xdf,0x19,0xdb,0x27,0xb9,0x9d,0xbf,0x0a,0x89,0x43,0xee,
 0x6a,0x53,0x95,0xb4,0x86,0x64,0xe3,0x93,0x0b,0xdb,0x0c,0xf5,
 0xa4,0xe7,0x96,0x46,0x52,0xe2,0xce,0xaf,0xae,0xe3,0x64,0x5e,
 0xe6,0x68,0xa0,0x47,0xf3,0xfb,0x25,0x9a,0xff,0xac,0x4e,0xae,
 0xc7,0x60,0x14,0xc2,0xa0,0x1d,0x20,0x64,0x9e,0x96,0xa1,0x9a,
 0xbd,0xe0,0xcd,0x22,
};
```

B.4.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
    #Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjUyMjNF
    #MENEQzQ1RDY1NzU2NjhENjRDNTUyMDA0MTI0IiwgImciOiAiQjQ3QjJDM0UzMjhFQT
    #lGMzE4Q0E2OTVBRjY1REM4MzhCQTVBNThCREI4MDg3QUU2QzZBQjQ4NkE4QkZDQjEz
    #RDM1QjMwMURERENBOTFEMTU0MEE0MTA2RDM4QkExQkYxNTJFNUJCQTEwNDE5M0M3RS
    #IsICJ5YSI6ICIzM0Q1NjFGMTNDRkMwRENBMjc5QzMwRThDREU4OTUxNzVEQzI1NDgz
    #ODkyODE5RUJBMTMyRDU4QzEzQzA0NjJBOEVCMEQ3M0ZEQTk0MTk1MDU5NEJFRjUxOT
    #FEODM5NDY5MUY4NkVERkZDQUQ2QzFFIiwgIkFEYSI6ICI0MTQ0NjEiLCAiWWEiOiAi
    #RDI5MjZBREY3MzdEOERGRTQ2QUU0N0NERDBBNTZCQkNEQUQ5Njk1ODUxMkQzOUY5OD
    #Y1QUQxQjY0QjUzMDRENEQxNjk2OTgzQ0Q2RDM4RDU3RTNDREIxMDgwMjYyMTk4QjVE
    #NEM0OTdDNjQ5MjZBNSIsICJ5YiI6ICIyNTIzQzk2OUY2OEZBMkIyQUVBMjk0QzI1Mz
    #lFRjM2RUIxRTA1NThBQkQxNDcxMkE3ODI4RjE2QTg1RUQyQzdFNzdFMkJERDQxODk5
    #NDQwNUZCMUI1N0I2QkJBQURENjY4NDk4OTJBQUM5RDgxNDAyIiwgIkFEYiI6ICI0MT
    #Q0NjIiLCAiWWIiOiAiMkU2MjBFNUQwQkM0NjM3RjFGRjk3RERDQ0IwMTUxNDFCMkMz
    #MjQxQjRBQkNCNTkzMUE3RTIyQTk5RkUyNTZBRTU0MzczM0NFNEU0QzQxODI2RERBMT
    #QyRkJGRUIyNjM5NzI3QTUyMzNGOEYwQTQyQSIsICJLIjogIkVFMEMwRUFEMDc1RUQ1
    #NjM3RTE2N0NEM0UxNjQxNUMxMTRBRUU2RTIxQURBRERDMERFOTE1MUIyOTMwODMzRT
    #I4NjJGNEU0RUMzMEM5MURFNUM1MjQ5QTcyNEQ5MTE0ODYwRjJCQ0JCQjZDQTgzRkQi
    #LCAiSVNLX0lSIjogIjZGM0E2NkNFMjBFMUJBRDI3OUQzMkQ2NTg0NENENjZCNzM1Mk
    #VCNjFBOUVCMzc2MzExNjhGRTgwM0M4M0M2NjAxQzZBM0UwODY3RjRCMkRCRTJCMENC
    #MUI1MzA5MTI4OTJEMkI1RUI5QUYyRTNBQjNEODY4MkEzM0UxNzhEMUQyIiwgIklTS1
    #9TWSI6ICI1OTAxMkIzMEI3QTQwNTBEMUU4M0Y3MTU3REE2Q0RENUJBRjc3QkY0MzJG
    #NTQ1RTdCQUExQTJBMTBCQzI5RUYxMzQ1RjlDRkUyRkI0NDhCRkU4QTQ4QTQ3MEQ5ME
    #Q5NENENzNFNEM5OUZDMUM4QjY1MUQ0M0ZCM0VCRUQzQTg4RiIsICJzaWRfb3V0cHV0
    #X2lyIjogIjlCM0ZCOTM2RTJCOUQwMERBNzA4NDEzOTJCRjAxMkYzRDBDMUJDRjBBOD
    #g2MTAyMEY3ODgyQjBEN0YxRTU0RUFGNkU0RDg1MEVBNEVEMzE2OUU2MzBGMjk2OEQ2
    #OUUxOTMzOEFFMjNENzRGRUJGMDAzQjQxNjAwQjBERTkxQUQ2IiwgInNpZF9vdXRwdX
    #Rfb2MiOiAiREZERjE5REIyN0I5OURCRjBBODk0M0VFNkE1Mzk1QjQ4NjY0RTM5MzBC
    #REIwQ0Y1QTRFNzk2NDY1MkUyQ0VBRkFFRTM2NDVFRTY2OEEwNDdGM0ZCMjU5QUZGQU
    #M0RUFFQzc2MDE0QzJBMDFEMjA2NDlFOTZBMTlBQkRFMENEMjIifQ==
```

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'\x0bA_initiator\x0bB_responder'
       CI = 0b415f696e69746961746f720b425f726573706f6e646572
       sid = 34b36454cab2e7842c389f7d88ecb7df
     Outputs
       generator_string(PRS,G.DSI,CI,sid,H.s_in_bytes):
       (length: 106 bytes)
         1e435061636550323536 5f584d443a5348412d3235365f535357555f
         4e555f0850617373776f726417000000000000000000000000000000
         0000000000000000180b415f696e69746961746f720b425f72657370
         6f6e6465721034b36454cab2e7842c389f7d88ecb7df
       generator g: (length: 65 bytes)
         0439bff2b051701594d3e9c7e93be9213af15db42214dfc4f7ee929a
         6697f774d7ba5fb289b982399e4acd281a988bf058ea7ff6d7ff34fd
         b72157bb464ff4af87
```

B.5.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS0yNTYiLCAiSC5zX2luX2J5dGVzIjogNjQsICJQUlMiOiAiNTA2MT
#czNzM3NzZGNzI2NCIsICJaUEFEIGxlbmd0aCI6IDIzLCAiRFNJIjogIjQzNTA2MTYz
#NjU1MDMyMzUzNjVGNTg0RDQ0M0E1MzQ4NDEyRDMyMzUzNjVGNTM1MzU3NTU1RjRFNT
#U1RiIsICJDSSI6ICIwQjQxNUY2OTZFNjk3NDY5NjE3NDZGNzIwQjQyNUY3MjY1NzM3
#MDZGNkU2NDY1NzIiLCAic2lkIjogIjM0QjM2NDU0Q0FCMkU3ODQyQzM4OUY3RDg4RU
#NCN0RGIiwgImdlbmVyYXRvcl9zdHJpbmcoRy5EU0ksUFJTLENJLHNpZCxILnNfaW5f
#Ynl0ZXMpIjogIjFFNDM1MDYxNjM2NTUwMzIzNTM2NUY1ODRENDQzQTUzNDg0MTJEMz
#IzNTM2NUY1MzUzNTc1NTVGNEU1NTVGMDg1MDYxNzM3Mzc3NkY3MjY0MTcwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMTgwQjQxNUY2OT
#ZFNjk3NDY5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIxMDM0QjM2NDU0
#Q0FCMkU3ODQyQzM4OUY3RDg4RUNCN0RGIiwgImdlbmVyYXRvciBnIjogIjA0MzlCRk
#YyQjA1MTcwMTU5NEQzRTlDN0U5M0JFOTIxM0FGMTVEQjQyMjE0REZDNEY3RUU5MjlB
#NjY5N0Y3NzREN0JBNUZCMjg5Qjk4MjM5OUU0QUNEMjgxQTk4OEJGMDU4RUE3RkY2RD
#dGRjM0RkRCNzIxNTdCQjQ2NEZGNEFGODcifQ==
```

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)
         04cf55f0a53a1b4c43002e1be8171f42737cf20b7cd6177b901ef962
         c2e2d486b2f738263c6da5aa902fe185ae2cda587df8d27a16fc19c3
         2a7b31aab097919736
       Alternative correct value for Ya: (-ya)*g:
       (length: 65 bytes)
         04cf55f0a53a1b4c43002e1be8171f42737cf20b7cd6177b901ef962
         c2e2d486b208c7d9c2925a5570d01e7a51d325a782072d85ea03e63c
         d584ce554f686e68c9
```

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)
         046f538f9b8eb4e628bcace0fbba7d36fea44e98334d233c22101a2f
         28eb3afc1a61b2bd14c0f11b222b5ea34df7756d104e2b5da09c9e8f
         fa330c150f47e84cef
       Alternative correct value for Yb: (-yb)*g:
       (length: 65 bytes)
         046f538f9b8eb4e628bcace0fbba7d36fea44e98334d233c22101a2f
         28eb3afc1a9e4d42ea3f0ee4ded4a15cb2088a92efb1d4a260636170
         05ccf3eaf0b817b310
```

B.5.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 32 bytes)
         9dd152b687ded1e071cc0625bd1ffff4c4ccd7d77cb4987c7d1e4ecb
         3a0db812
       scalar_mult_vfy(yb,Ya): (length: 32 bytes)
         9dd152b687ded1e071cc0625bd1ffff4c4ccd7d77cb4987c7d1e4ecb
         3a0db812
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 140 bytes)
  4104cf55f0a53a1b4c43002e1be8171f42737cf20b7cd6177b901ef9
  62c2e2d486b2f738263c6da5aa902fe185ae2cda587df8d27a16fc19
  c32a7b31aab0979197360341446141046f538f9b8eb4e628bcace0fb
  ba7d36fea44e98334d233c22101a2f28eb3afc1a61b2bd14c0f11b22
  2b5ea34df7756d104e2b5da09c9e8ffa330c150f47e84cef03414462
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
  4e555f5f49534b1034b36454cab2e7842c389f7d88ecb7df209dd152
  b687ded1e071cc0625bd1ffff4c4ccd7d77cb4987c7d1e4ecb3a0db8
  124104cf55f0a53a1b4c43002e1be8171f42737cf20b7cd6177b901e
  f962c2e2d486b2f738263c6da5aa902fe185ae2cda587df8d27a16fc
  19c32a7b31aab0979197360341446141046f538f9b8eb4e628bcace0
  fbba7d36fea44e98334d233c22101a2f28eb3afc1a61b2bd14c0f11b
  222b5ea34df7756d104e2b5da09c9e8ffa330c150f47e84cef034144
  62
ISK result: (length: 32 bytes)
  d67704f1c69b85736f273e73198a79fe5e4f60cb405e32f708e0aff5
  fdb5f9db
```

B.5.6. Test vector for ISK calculation parallel execution

```
       transcript_oc(Ya,ADa,Yb,ADb): (length: 142 bytes)
         6f634104cf55f0a53a1b4c43002e1be8171f42737cf20b7cd6177b90
         1ef962c2e2d486b2f738263c6da5aa902fe185ae2cda587df8d27a16
         fc19c32a7b31aab0979197360341446141046f538f9b8eb4e628bcac
         e0fbba7d36fea44e98334d233c22101a2f28eb3afc1a61b2bd14c0f1
         1b222b5ea34df7756d104e2b5da09c9e8ffa330c150f47e84cef0341
         4462
       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)
         22435061636550323536 5f584d443a5348412d3235365f535357555f
         4e555f5f49534b1034b36454cab2e7842c389f7d88ecb7df209dd152
         b687ded1e071cc0625bd1ffff4c4ccd7d77cb4987c7d1e4ecb3a0db8
         126f634104cf55f0a53a1b4c43002e1be8171f42737cf20b7cd6177b
         901ef962c2e2d486b2f738263c6da5aa902fe185ae2cda587df8d27a
         16fc19c32a7b31aab0979197360341446141046f538f9b8eb4e628bc
         ace0fbba7d36fea44e98334d233c22101a2f28eb3afc1a61b2bd14c0
         f11b222b5ea34df7756d104e2b5da09c9e8ffa330c150f47e84cef03
         414462
       ISK result: (length: 32 bytes)
         724ffa2de3685f12a10bb2eb013cb6c0378acef40b78f5b21dbd69f7
         2478e32b
```

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

```
       H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
       (length: 32 bytes)
         28e98006db715c2de9817e847f937336ba510ed288cd55a6e7b020d0
         c85e74fe
       H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
       (length: 32 bytes)
         dc11fd7db69958c3bd30d72171377444261975c599fa6e85f73bac00
         a85ad5c3
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
    0x04,0x39,0xbf,0xf2,0xb0,0x51,0x70,0x15,0x94,0xd3,0xe9,0xc7,
    0xe9,0x3b,0xe9,0x21,0x3a,0xf1,0x5d,0xb4,0x22,0x14,0xdf,0xc4,
    0xf7,0xee,0x92,0x9a,0x66,0x97,0xf7,0x74,0xd7,0xba,0x5f,0xb2,
    0x89,0xb9,0x82,0x39,0x9e,0x4a,0xcd,0x28,0x1a,0x98,0x8b,0xf0,
    0x58,0xea,0x7f,0xf6,0xd7,0xff,0x34,0xfd,0xb7,0x21,0x57,0xbb,
    0x46,0x4f,0xf4,0xaf,0x87,
   };
   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[] = {
    0x04,0xcf,0x55,0xf0,0xa5,0x3a,0x1b,0x4c,0x43,0x00,0x2e,0x1b,
    0xe8,0x17,0x1f,0x42,0x73,0x7c,0xf2,0x0b,0x7c,0xd6,0x17,0x7b,
    0x90,0x1e,0xf9,0x62,0xc2,0xe2,0xd4,0x86,0xb2,0xf7,0x38,0x26,
    0x3c,0x6d,0xa5,0xaa,0x90,0x2f,0xe1,0x85,0xae,0x2c,0xda,0x58,
    0x7d,0xf8,0xd2,0x7a,0x16,0xfc,0x19,0xc3,0x2a,0x7b,0x31,0xaa,
    0xb0,0x97,0x91,0x97,0x36,
   };
   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[] = {
    0x04,0x6f,0x53,0x8f,0x9b,0x8e,0xb4,0xe6,0x28,0xbc,0xac,0xe0,
    0xfb,0xba,0x7d,0x36,0xfe,0xa4,0x4e,0x98,0x33,0x4d,0x23,0x3c,
    0x22,0x10,0x1a,0x2f,0x28,0xeb,0x3a,0xfc,0x1a,0x61,0xb2,0xbd,
    0x14,0xc0,0xf1,0x1b,0x22,0x2b,0x5e,0xa3,0x4d,0xf7,0x75,0x6d,
    0x10,0x4e,0x2b,0x5d,0xa0,0x9c,0x9e,0x8f,0xfa,0x33,0x0c,0x15,
    0x0f,0x47,0xe8,0x4c,0xef,
   };
   const unsigned char tc_K[] = {
    0x9d,0xd1,0x52,0xb6,0x87,0xde,0xd1,0xe0,0x71,0xcc,0x06,0x25,
    0xbd,0x1f,0xff,0xf4,0xc4,0xcc,0xd7,0xd7,0x7c,0xb4,0x98,0x7c,
    0x7d,0x1e,0x4e,0xcb,0x3a,0x0d,0xb8,0x12,
   };
   const unsigned char tc_ISK_IR[] = {
```

```
    0xd6,0x77,0x04,0xf1,0xc6,0x9b,0x85,0x73,0x6f,0x27,0x3e,0x73,
    0x19,0x8a,0x79,0xfe,0x5e,0x4f,0x60,0xcb,0x40,0x5e,0x32,0xf7,
    0x08,0xe0,0xaf,0xf5,0xfd,0xb5,0xf9,0xdb,
   };
   const unsigned char tc_ISK_SY[] = {
    0x72,0x4f,0xfa,0x2d,0xe3,0x68,0x5f,0x12,0xa1,0x0b,0xb2,0xeb,
    0x01,0x3c,0xb6,0xc0,0x37,0x8a,0xce,0xf4,0x0b,0x78,0xf5,0xb2,
    0x1d,0xbd,0x69,0xf7,0x24,0x78,0xe3,0x2b,
   };
   const unsigned char tc_sid_out_ir[] = {
    0x28,0xe9,0x80,0x06,0xdb,0x71,0x5c,0x2d,0xe9,0x81,0x7e,0x84,
    0x7f,0x93,0x73,0x36,0xba,0x51,0x0e,0xd2,0x88,0xcd,0x55,0xa6,
    0xe7,0xb0,0x20,0xd0,0xc8,0x5e,0x74,0xfe,
   };
   const unsigned char tc_sid_out_oc[] = {
    0xdc,0x11,0xfd,0x7d,0xb6,0x99,0x58,0xc3,0xbd,0x30,0xd7,0x21,
    0x71,0x37,0x74,0x44,0x26,0x19,0x75,0xc5,0x99,0xfa,0x6e,0x85,
    0xf7,0x3b,0xac,0x00,0xa8,0x5a,0xd5,0xc3,
   };
```

B.5.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
    #Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjM0QjM2
    #NDU0Q0FCMkU3ODQyQzM4OUY3RDg4RUNCN0RGIiwgImciOiAiMDQzOUJGRjJCMDUxNz
    #AxNTk0RDNFOUM3RTkzQkU5MjEzQUYxNURCNDIyMTRERkM0RjdFRTkyOUE2Njk3Rjc3
    #NEQ3QkE1RkIyODlCOTgyMzk5RTRBQ0QyODFBOTg4QkYwNThFQTdGRjZEN0ZGMzRGRE
    #I3MjE1N0JCNDY0RkY0QUY4NyIsICJ5YSI6ICIzNzU3NENGQkYxQjk1RkY2QThFMkQ3
    #QkU0NjJENEQwMUU2RERFMjYxOEYzNEY0REU5REY4NjlCMjRGNTMyQzVEIiwgIkFEYS
    #I6ICI0MTQ0NjEiLCAiWWEiOiAiMDRDRjU1RjBBNTNBMUI0QzQzMDAyRTFCRTgxNzFG
    #NDI3MzdDRjIwQjdDRDYxNzdCOTAxRUY5NjJDMkUyRDQ4NkIyRjczODI2M0M2REE1QU
    #E5MDJGRTE4NUFFMkNEQTU4N0RGOEQyN0ExNkZDMTlDMzJBN0IzMUFBQjA5NzkxOTcz
    #NiIsICJ5YiI6ICJFNTY3MkZDOUVCNEU3MjFGNDFEODAxODFFQzRDOUZEOTg4NjY2OE
    #FDQzQ4MDI0RDMzQzgyQkIxMDJBRUNCQTUyIiwgIkFEYiI6ICI0MTQ0NjIiLCAiWWIi
    #OiAiMDQ2RjUzOEY5QjhFQjRFNjI4QkNBQ0UwRkJCQTdEMzZGRUE0NEU5ODMzNEQyMz
    #NDMjIxMDFBMkYyOEVCM0FGQzFBNjFCMkJEMTRDMEYxMUIyMjJCNUVBMzRERjc3NTZE
    #MTA0RTJCNURBMDlDOUU4RkZBMzMwQzE1MEY0N0U4NENFRiIsICJLIjogIjlERDE1Mk
    #I2ODdERUQxRTA3MUNDMDYyNUJEMUZGRkY0QzRDQ0Q3RDc3Q0I0OTg3QzdEMUU0RUNC
    #M0EwREI4MTIiLCAiSVNLX0lSIjogIkQ2NzcwNEYxQzY5Qjg1NzM2RjI3M0U3MzE5OE
    #E3OUZFNUU0RjYwQ0I0MDVFMzJGNzA4RTBBRkY1RkRCNUY5REIiLCAiSVNLX1NZIjog
    #IjcyNEZGQTJERTM2ODVGMTJBMTBCQjJFQjAxM0NCNkMwMzc4QUNFRjQwQjc4RjVCMj
    #FEQkQ2OUY3MjQ3OEUzMkIiLCAic2lkX291dHB1dF9pciI6ICIyOEU5ODAwNkRCNzE1
    #QzJERTk4MTdFODQ3RjkzNzMzNkJBNTEwRUQyODhDRDU1QTZFN0IwMjBEMEM4NUU3NE
    #ZFIiwgInNpZF9vdXRwdXRfb2MiOiAiREMxMUZEN0RCNjk5NThDM0JEMzBENzIxNzEz
    #Nzc0NDQyNjE5NzVDNTk5RkE2RTg1RjczQkFDMDBBODVBRDVDMyJ9
```

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'\x0bA_initiator\x0bB_responder'
     CI = 0b415f696e69746961746f720b425f726573706f6e646572
     sid = 5b3773aa90e8f23c61563a4b645b276c
   Outputs
     generator_string(PRS,G.DSI,CI,sid,H.s_in_bytes):
     (length: 170 bytes)
       1e435061636550333834 5f584d443a5348412d3338345f535357555f
       4e555f0850617373776f726457000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000
       0000000000000000000000000180b415f696e69746961746f
       720b425f726573706f6e646572105b3773aa90e8f23c61563a4b645b
       276c
     generator g: (length: 97 bytes)
       04d938e00eaa8d44fcac33dd5d10404f0c57e0fb1dfdcca3fae47a6a
       14bbe52a6ac82a856d9de14e0223a1e1a5bd3fb38be951af748723f8
       f64835217ad5bc05a4fc0e1d7c17cf14c3bda93d05cef96b86483fc5
       f51570fc27f0a3c19cd2ece2a2
```

B.6.1.1. Testvectors as JSON file encoded as BASE64

```
    #eyJIIjogIlNIQS0zODQiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
    #E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiA4NywgIkRTSSI6ICI0MzUwNjE2
    #MzY1NTAzMzM4MzQ1RjU4NEQ0NDNBNTM0ODQxMkQzMzM4MzQ1RjUzNTM1NzU1NUY0RT
    #U1NUYiLCAiQ0kiOiAiMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyMEI0MjVGNzI2NTcz
    #NzA2RjZFNjQ2NTcyIiwgInNpZCI6ICI1QjM3NzNBQTkwRThGMjNDNjE1NjNBNEI2ND
    #VCMjc2QyIsICJnZW5lcmF0b3Jfc3RyaW5nKEcuRFNJLFBSUyxDSSxzaWQsSC5zX2lu
    #X2J5dGVzKSI6ICIxRTQzNTA2MTYzNjU1MDMzMzgzNDVGNTg0RDQ0M0E1MzQ4NDEyRD
    #MzMzgzNDVGNTM1MzU3NTU1RjRFNTU1RjA4NTA2MTczNzM3NzZGNzI2NDU3MDAwMDAw
    #MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
    #MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
    #AwMDAwMDAwMDAwMDAwMDAwMTgwQjQxNUY2OTZFNjk3NDY5NjE3NDZGNzIwQjQy
    #NUY3MjY1NzM3MDZGNkU2NDY1NzIxMDVCMzc3M0FBOTBFOEYyM0M2MTU2M0E0QjY0NU
    #IyNzZDIiwgImdlbmVyYXRvciBnIjogIjA0RDkzOEUwMEVBQThENDRGQ0FDMzNERDVE
    #MTA0MDRGMEM1N0UwRkIxREZEQ0NBM0ZBRTQ3QTZBMTRCQkU1MkE2QUM4MkE4NTZEOU
    #RFMTRFMDIyM0ExRTFBNUJEM0ZCMzhCRTk1MUFGNzQ4NzIzRjhGNjQ4MzUyMTdBRDVC
    #QzA1QTRGQzBFMUQ3QzE3Q0YxNEMzQkRBOTNEMDVDRUY5NkI4NjQ4M0ZDNUY1MTU3ME
    #ZDMjdGMEEzQzE5Q0QyRUNFMkEyIn0=
```

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)
    043973faff49d85b1a14f114d0550a8ec0e5defcc7399c2e19003251
    b1ddb9e786489b66be3cf8601880bd426a8a52f96968229d35032ac9
    cc3ef6c1fd67ce1f7979bb1be9f4ff63e34fc6df42ebb481989596f6
    3c6bd987fffd2527bdfd6b05d6
  Alternative correct value for Ya: (-ya)*g:
  (length: 97 bytes)
    043973faff49d85b1a14f114d0550a8ec0e5defcc7399c2e19003251
    b1ddb9e786489b66be3cf8601880bd426a8a52f96997dd62cafcd536
    33c1093e029831e0868644e4160b009c1cb03920bd144b7e666a6909
    c29426780002dad8430294fa29
```

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)
         04a94b80cc266671a00ef719ee249bda79cbf66788784611e9c18835
         2ac7dbf23f730a4c3ea320e7a27a657ac336e7ee9c9bf49a5dcb715b
         b9b2ac2e5433bba61b2b186b7d6a539015096f351b36df579469d50a
         2f7ca1c615bd7e28633e5faefb
       Alternative correct value for Yb: (-yb)*g:
       (length: 97 bytes)
         04a94b80cc266671a00ef719ee249bda79cbf66788784611e9c18835
         2ac7dbf23f730a4c3ea320e7a27a657ac336e7ee9c640b65a2348ea4
         464d53d1abcc4459e4d4e7948295ac6feaf690cae4c920a86a962af5
         cf835e39ea4281d79dc1a05104
```

B.6.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 48 bytes)
         03ad91b03f8312fdd6aa1986dafc684adb2d93a1248e99a22843c502
         397903b464768f57d2ef19fda00542dc7636031f
       scalar_mult_vfy(yb,Ya): (length: 48 bytes)
         03ad91b03f8312fdd6aa1986dafc684adb2d93a1248e99a22843c502
         397903b464768f57d2ef19fda00542dc7636031f
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 204 bytes)
  61043973faff49d85b1a14f114d0550a8ec0e5defcc7399c2e190032
  51b1ddb9e786489b66be3cf8601880bd426a8a52f96968229d35032a
  c9cc3ef6c1fd67ce1f7979bb1be9f4ff63e34fc6df42ebb481989596
  f63c6bd987fffd2527bdfd6b05d6034144616104a94b80cc266671a0
  0ef719ee249bda79cbf66788784611e9c188352ac7dbf23f730a4c3e
  a320e7a27a657ac336e7ee9c9bf49a5dcb715bb9b2ac2e5433bba61b
  2b186b7d6a539015096f351b36df579469d50a2f7ca1c615bd7e2863
  3e5faefb03414462
DSI = G.DSI_ISK, b'CPaceP384_XMD:SHA-384_SSWU_NU__ISK':
(length: 34 bytes)
  43506163655033383345f584d443a5348412d3338345f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 305 bytes)
  224350616365503338345f584d443a5348412d3338345f535357555f
  4e555f5f49534b105b3773aa90e8f23c61563a4b645b276c3003ad91
  b03f8312fdd6aa1986dafc684adb2d93a1248e99a22843c502397903
  b464768f57d2ef19fda00542dc7636031f61043973faff49d85b1a14
  f114d0550a8ec0e5defcc7399c2e19003251b1ddb9e786489b66be3c
  f8601880bd426a8a52f96968229d35032ac9cc3ef6c1fd67ce1f7979
  bb1be9f4ff63e34fc6df42ebb481989596f63c6bd987fffd2527bdfd
  6b05d6034144616104a94b80cc266671a00ef719ee249bda79cbf667
  88784611e9c188352ac7dbf23f730a4c3ea320e7a27a657ac336e7ee
  9c9bf49a5dcb715bb9b2ac2e5433bba61b2b186b7d6a539015096f35
  1b36df579469d50a2f7ca1c615bd7e28633e5faefb03414462
ISK result: (length: 48 bytes)
  407d3e94916138a8a0cd3b6b64cdbd9a665fd83e706508649bc43b04
  7b73cf27017d2cc74533f18a5075a6748de5a026
```

B.6.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 206 bytes)
  6f636104a94b80cc266671a00ef719ee249bda79cbf66788784611e9
  c188352ac7dbf23f730a4c3ea320e7a27a657ac336e7ee9c9bf49a5d
  cb715bb9b2ac2e5433bba61b2b186b7d6a539015096f351b36df5794
  69d50a2f7ca1c615bd7e28633e5faefb0341446261043973faff49d8
  5b1a14f114d0550a8ec0e5defcc7399c2e19003251b1ddb9e786489b
  66be3cf8601880bd426a8a52f96968229d35032ac9cc3ef6c1fd67ce
  1f7979bb1be9f4ff63e34fc6df42ebb481989596f63c6bd987fffd25
  27bdfd6b05d603414461
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
  4e555f5f49534b105b3773aa90e8f23c61563a4b645b276c3003ad91
  b03f8312fdd6aa1986dafc684adb2d93a1248e99a22843c502397903
  b464768f57d2ef19fda00542dc7636031f6f636104a94b80cc266671
  a00ef719ee249bda79cbf66788784611e9c188352ac7dbf23f730a4c
  3ea320e7a27a657ac336e7ee9c9bf49a5dcb715bb9b2ac2e5433bba6
  1b2b186b7d6a539015096f351b36df579469d50a2f7ca1c615bd7e28
  633e5faefb0341446261043973faff49d85b1a14f114d0550a8ec0e5
  defcc7399c2e19003251b1ddb9e786489b66be3cf8601880bd426a8a
  52f96968229d35032ac9cc3ef6c1fd67ce1f7979bb1be9f4ff63e34f
  c6df42ebb481989596f63c6bd987fffd2527bdfd6b05d603414461
ISK result: (length: 48 bytes)
  b2c28ddad4a77680469bd090bad8098d1c22e7050553acdacd13614d
  eed72bb7821c50ddc4b30b4b54151e07077d1985
```

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

```
H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
(length: 48 bytes)
  ad0247e90c1124ce6da6f62d55daf30fada0b08160164bb93ccd5a2d
  f9750b91a4517184872e2bcfb4fff83bbad02371
H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
(length: 48 bytes)
  4c7b6523ae4c7a5df9d709b59dd1b033152040715391c7ed7c60a12c
  c3f40e251fbda5b78c63bc465dac5ca6046f3fc0
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
    0x04,0xd9,0x38,0xe0,0x0e,0xaa,0x8d,0x44,0xfc,0xac,0x33,0xdd,
    0x5d,0x10,0x40,0x4f,0x0c,0x57,0xe0,0xfb,0x1d,0xfd,0xcc,0xa3,
    0xfa,0xe4,0x7a,0x6a,0x14,0xbb,0xe5,0x2a,0x6a,0xc8,0x2a,0x85,
    0x6d,0x9d,0xe1,0x4e,0x02,0x23,0xa1,0xe1,0xa5,0xbd,0x3f,0xb3,
    0x8b,0xe9,0x51,0xaf,0x74,0x87,0x23,0xf8,0xf6,0x48,0x35,0x21,
    0x7a,0xd5,0xbc,0x05,0xa4,0xfc,0x0e,0x1d,0x7c,0x17,0xcf,0x14,
    0xc3,0xbd,0xa9,0x3d,0x05,0xce,0xf9,0x6b,0x86,0x48,0x3f,0xc5,
    0xf5,0x15,0x70,0xfc,0x27,0xf0,0xa3,0xc1,0x9c,0xd2,0xec,0xe2,
    0xa2,
   };
   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[] = {
    0x04,0x39,0x73,0xfa,0xff,0x49,0xd8,0x5b,0x1a,0x14,0xf1,0x14,
    0xd0,0x55,0x0a,0x8e,0xc0,0xe5,0xde,0xfc,0xc7,0x39,0x9c,0x2e,
    0x19,0x00,0x32,0x51,0xb1,0xdd,0xb9,0xe7,0x86,0x48,0x9b,0x66,
    0xbe,0x3c,0xf8,0x60,0x18,0x80,0xbd,0x42,0x6a,0x8a,0x52,0xf9,
    0x69,0x68,0x22,0x9d,0x35,0x03,0x2a,0xc9,0xcc,0x3e,0xf6,0xc1,
    0xfd,0x67,0xce,0x1f,0x79,0x79,0xbb,0x1b,0xe9,0xf4,0xff,0x63,
    0xe3,0x4f,0xc6,0xdf,0x42,0xeb,0xb4,0x81,0x98,0x95,0x96,0xf6,
    0x3c,0x6b,0xd9,0x87,0xff,0xfd,0x25,0x27,0xbd,0xfd,0x6b,0x05,
    0xd6,
   };
   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[] = {
    0x04,0xa9,0x4b,0x80,0xcc,0x26,0x66,0x71,0xa0,0x0e,0xf7,0x19,
    0xee,0x24,0x9b,0xda,0x79,0xcb,0xf6,0x67,0x88,0x78,0x46,0x11,
    0xe9,0xc1,0x88,0x35,0x2a,0xc7,0xdb,0xf2,0x3f,0x73,0x0a,0x4c,
    0x3e,0xa3,0x20,0xe7,0xa2,0x7a,0x65,0x7a,0xc3,0x36,0xe7,0xee,
    0x9c,0x9b,0xf4,0x9a,0x5d,0xcb,0x71,0x5b,0xb9,0xb2,0xac,0x2e,
    0x54,0x33,0xbb,0xa6,0x1b,0x2b,0x18,0x6b,0x7d,0x6a,0x53,0x90,
    0x15,0x09,0x6f,0x35,0x1b,0x36,0xdf,0x57,0x94,0x69,0xd5,0x0a,
    0x2f,0x7c,0xa1,0xc6,0x15,0xbd,0x7e,0x28,0x63,0x3e,0x5f,0xae,
    0xfb,
   };
   const unsigned char tc_K[] = {
    0x03,0xad,0x91,0xb0,0x3f,0x83,0x12,0xfd,0xd6,0xaa,0x19,0x86,
    0xda,0xfc,0x68,0x4a,0xdb,0x2d,0x93,0xa1,0x24,0x8e,0x99,0xa2,
    0x28,0x43,0xc5,0x02,0x39,0x79,0x03,0xb4,0x64,0x76,0x8f,0x57,
    0xd2,0xef,0x19,0xfd,0xa0,0x05,0x42,0xdc,0x76,0x36,0x03,0x1f,
   };
   const unsigned char tc_ISK_IR[] = {
    0x40,0x7d,0x3e,0x94,0x91,0x61,0x38,0xa8,0xa0,0xcd,0x3b,0x6b,
    0x64,0xcd,0xbd,0x9a,0x66,0x5f,0xd8,0x3e,0x70,0x65,0x08,0x64,
    0x9b,0xc4,0x3b,0x04,0x7b,0x73,0xcf,0x27,0x01,0x7d,0x2c,0xc7,
    0x45,0x33,0xf1,0x8a,0x50,0x75,0xa6,0x74,0x8d,0xe5,0xa0,0x26,
   };
   const unsigned char tc_ISK_SY[] = {
    0xb2,0xc2,0x8d,0xda,0xd4,0xa7,0x76,0x80,0x46,0x9b,0xd0,0x90,
    0xba,0xd8,0x09,0x8d,0x1c,0x22,0xe7,0x05,0x05,0x53,0xac,0xda,
    0xcd,0x13,0x61,0x4d,0xee,0xd7,0x2b,0xb7,0x82,0x1c,0x50,0xdd,
    0xc4,0xb3,0x0b,0x4b,0x54,0x15,0x1e,0x07,0x07,0x7d,0x19,0x85,
   };
   const unsigned char tc_sid_out_ir[] = {
    0xad,0x02,0x47,0xe9,0x0c,0x11,0x24,0xce,0x6d,0xa6,0xf6,0x2d,
    0x55,0xda,0xf3,0x0f,0xad,0xa0,0xb0,0x81,0x60,0x16,0x4b,0xb9,
    0x3c,0xcd,0x5a,0x2d,0xf9,0x75,0x0b,0x91,0xa4,0x51,0x71,0x84,
    0x87,0x2e,0x2b,0xcf,0xb4,0xff,0xf8,0x3b,0xba,0xd0,0x23,0x71,
   };
   const unsigned char tc_sid_out_oc[] = {
    0x4c,0x7b,0x65,0x23,0xae,0x4c,0x7a,0x5d,0xf9,0xd7,0x09,0xb5,
    0x9d,0xd1,0xb0,0x33,0x15,0x20,0x40,0x71,0x53,0x91,0xc7,0xed,
    0x7c,0x60,0xa1,0x2c,0xc3,0xf4,0x0e,0x25,0x1f,0xbd,0xa5,0xb7,
    0x8c,0x63,0xbc,0x46,0x5d,0xac,0x5c,0xa6,0x04,0x6f,0x3f,0xc0,
   };
```

B.6.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
    #Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjVCMzc3
    #M0FBOTBFOEYyM0M2MTU2M0E0QjY0NUIyNzZDIiwgImciOiAiMDREOTM4RTAwRUFBOE
    #Q0NEZDQUMzM0RENUQxMDQwNEYwQzU3RTBGQjFERkRDQ0EzRkFFNDdBNkExNEJCRTUy
    #QTZBQzgyQTg1NkQ5REUxNEUwMjIzQTFFMUE1QkQzRkIzOEJFOTUxQUY3NDg3MjNGOE
    #Y2NDgzNTIxN0FENUJDMDVBNEZDMEUxRDdDMTdDRjE0QzNCREE5M0QwNUNFRjk2Qjg2
    #NDgzRkM1RjUxNTcwRkMyN0YwQTNDMTlDRDJFQ0UyQTIiLCAieWEiOiAiRUY0MzNERD
    #VBRDE0MkM4NjBFN0NCNjQwMEREMzE1RDM4OEQ1RUM1NDIwQzU1MEU5RDZGMDkwN0Yz
    #NzVEOTg4QkM0RDcwNDgzN0U0MzU2MUM0OTdFN0REOTNFRENEQjlEIiwgIkFEYSI6IC
    #I0MTQ0NjEiLCAiWWEiOiAiMDQzOTczRkFGRjQ5RDg1QjFBMTRGMTE0RDA1NTBBOEVD
    #MEU1REVGQ0M3Mzk5QzJFMTkwMDMyNTFCMUREQjlFNzg2NDg5QjY2QkUzQ0Y4NjAxOD
    #gwQkQ0MjZBOEE1MkY5Njk2ODIyOUQzNTAzMkFDOUNDM0VGNkMxRkQ2N0NFMUY3OTc5
    #QkIxQkU5RjRGRjYzRTM0RkM2REY0MkVCQjQ4MTk4OTU5NkY2M0M2QkQ5ODdGRkZEMj
    #UyN0JERkQ2QjA1RDYiLCAieWIiOiAiNTBCMEUzNkI5NUEyRURGQUE4MzQyQjg0M0RE
    #REM5MEIxNzUzMzBGMjM5OUMxQjM2NTg2REVEREEzQzI1NTk3NUYzMEJFNkE3NTBGOT
    #QwNEZDQ0M2MkE2MzIzQjVFNDcxIiwgIkFEYiI6ICI0MTQ0NjIiLCAiWWIiOiAiMDRB
    #OTRCODBDQzI2NjY3MUEwMEVGNzE5RUUyNDlCREE3OUNCRjY2Nzg4Nzg0NjExRTlDMT
    #g4MzUyQUM3REJGMjNGNzMwQTRDM0VBMzIwRTdBMjdBNjU3QUMzMzZFN0VFOUM5QkY0
    #OUE1RENCNzE1QkI5QjJBQzJFNTQzM0JCQTYxQjJCMTg2QjdENkE1MzkwMTUwOTZGMz
    #UxQjM2REY1Nzk0NjlENTBBMkY3Q0ExQzYxNUJEN0UyODYzM0U1RkFFRkIiLCAiSyI6
    #ICIwM0FEOTFCMDNGODMxMkZERDZBQTE5ODZEQUZDNjg0QURCMkQ5M0ExMjQ4RTk5QT
    #IyODQzQzUwMjM5NzkwM0I0NjQ3NjhGNTdEMkVGMTlGREEwMDU0MkRDNzYzNjAzMUYi
    #LCAiSVNLX0lSIjogIjQwN0QzRTk0OTE2MTM4QThBMENEM0I2QjY0Q0RCRDlBNjY1Rk
    #Q4M0U3MDY1MDg2NDlCQzQzQjA0N0I3M0NGMjcwMTdEMkNDNzQ1MzNGMThBNTA3NUE2
    #NzQ4REU1QTAyNiIsICJJU0tfU1kiOiAiQjJDMjhEREFENEE3NzY4MDQ2OUJEMDkwQk
    #FEODA5OEQxQzIyRTcwNTA1NTNBQ0RBQ0QxMzYxNERFRUQ3MkJCNzgyMUM1MEREQzRC
    #MzBCNEI1NDE1MUUwNzA3N0QxOTg1IiwgInNpZF9vdXRwdXRfaXIiOiAiQUQwMjQ3RT
    #kwQzExMjRDRTZEQTZGNjJENTVEQUYzMEZBREEwQjA4MTYwMTY0QkI5M0NDRDVBMkRG
    #OTc1MEI5MUE0NTE3MTg0ODcyRTJCQ0ZCNEZGRjgzQkJBRDAyMzcxIiwgInNpZF9vdX
    #RwdXRfb2MiOiAiNEM3QjY1MjNBRTRDN0E1REY5RDcwOUI1OUREMUIwMzMxNTIwNDA3
    #MTUzOTFDN0VEN0M2MEExMkNDM0Y0MEUyNTFGQkRBNUI3OEM2M0JDNDY1REFDNUNBNj
    #A0NkYzRkMwIn0=
```

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'\x0bA_initiator\x0bB_responder'
CI = 0b415f696e69746961746f720b425f726573706f6e646572
sid = 7e4b4791d6a8ef019b936c79fb7f2c57
```

Outputs

generator_string(PRS,G.DSI,CI,sid,H.s_in_bytes):
(length: 170 bytes)

```
1e435061636550353231 5f584d443a5348412d3531325f535357555f
4e555f0850617373776f726457000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000180b415f696e69746961746f
720b425f726573706f6e646572107e4b4791d6a8ef019b936c79fb7f
2c57
```

generator g: (length: 133 bytes)

```
0400aefb79f4ba40369bbe78a5c873b2ce75be111516deed28c5899a
8bbf8a96dbd28882afbc1501d8a4e9b25c172dfadcd96d10466f9c36
d23ea05c982df0bb1e353201c98da6ce14e3c424b6cf55e627c4f3c0
a4469e2aedaa692f5fe16d57baa8a9432aac320139e854d06bfb4daa
efc504ff2fe1d2c4f06443878a242a7b876b224040
```

B.7.1.1. Testvectors as JSON file encoded as BASE64

```
#eyJIIjogIlNIQS01MTIiLCAiSC5zX2luX2J5dGVzIjogMTI4LCAiUFJTIjogIjUwNj
#E3MzczNzc2RjcyNjQiLCAiWlBBRCBsZW5ndGgiOiA4NywgIkRTSSI6ICI0MzUwNjE2
#MzY1NTAzNTMyMzE1RjU4NEQ0NDNBNTM0ODQxMkQzNTMxMzI1RjUzNTM1NzU1NUY0RT
#U1NUYiLCAiQ0kiOiAiMEI0MTVGNjk2RTY5NzQ2OTYxNzQ2RjcyMEI0MjVGNzI2NTcz
#NzA2RjZFNjQ2NTcyIiwgInNpZCI6ICI3RTRCNDc5MUQ2QThFRjAxOUI5MzZDNzlGQj
#dGMkM1NyIsICJnZW5lcmF0b3Jfc3RyaW5nKEcuRFNJLFBSUyxDSSxzaWQsSC5zX2lu
#X2J5dGVzKSI6ICIxRTQzNTA2MTYzNjU1MDM1MzIzMTVGNTg0RDQ0M0E1MzQ4NDEyRD
#M1MzEzMjVGNTM1MzU3NTU1RjRFNTU1RjA4NTA2MTczNzM3NzZGNzI2NDU3MDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
#MDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMD
#AwMDAwMDAwMDAwMDAwMDAwMTgwQjQxNUY2OTZFNjk3NDY5NjE3NDZGNzIwQjQy
#NUY3MjY1NzM3MDZGNkU2NDY1NzIxMDdFNEI0NzkxRDZBOEVGMDE5QjkzNkM3OUZCN0
#YyQzU3IiwgImdlbmVyYXRvciBnIjogIjA0MDBBRUZCNzlGNEJBNDAzNjlCQkU3OEE1
#Qzg3M0IyQ0U3NUJFMTExNTE2REVFRDI4QzU4OTlBOEJCRjhBOTZEQkQyODg4MkFGQk
#MxNTAxRDhBNEU5QjI1QzE3MkRGQURDRDk2RDEwNDY2RjlDMzZEMjNFQTA1Qzk4MkRG
#MEJCMUUzNTMyMDFDOThEQTZDRTE0RTNDNDI0QjZDRjU1RTYyN0M0RjNDMEE0NDY5RT
#JBRURBQTY5MkY1RkUxNkQ1N0JBQThBOTQzMkFBQzMyMDEzOUU4NTREMDZCRkI0REFB
#RUZDNTA0RkYyRkUxRDJDNEYwNjQ0Mzg3OEEyNDJBN0I4NzZCMjI0MDQwIn0=
```

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)
    0400493381a239f5c33edc50f92a4ac7dbade7854c2ef090515567a5
    387c331fc9c7fdff4e151801a51ed20514412dd9832a0ebf4651e5d4
    cfebf1ebb89ca459a6311801d78c1142bf4eb46df664108bd05e8d97
    9f3bc13ea3e373ce07fd67e457f44927e0123d7566b527eb10ead212
    71d922f373fec78b4a3f4798f8fd52fb97859588de
  Alternative correct value for Ya: (-ya)*g:
  (length: 133 bytes)
    0400493381a239f5c33edc50f92a4ac7dbade7854c2ef090515567a5
    387c331fc9c7fdff4e151801a51ed20514412dd9832a0ebf4651e5d4
    cfebf1ebb89ca459a63118002873eebd40b14b92099bef742fa17268
    60c43ec15c1c8c31f802981ba80bb6d81fedc28a994ad814ef152ded
    8e26dd0c8c013874b5c0b8670702ad04687a6a7721
```

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)
         04012938f18169cf9f37048b973ec9ac06bb15eeb26d370957736775
         e6c5638153f00152fe632215eb22f407dc3a1a2c473f45751b6451bf
         626b781d8db61c537cb4310060575918b0c77f5208b31b89f6fc2cf8
         6bf494a1533a6f282ae0fc550133d51626c35989c01462aabab6dab9
         522cfb6a2ab214570f29560a7cc9619aa3e3341929
       Alternative correct value for Yb: (-yb)*g:
       (length: 133 bytes)
         04012938f18169cf9f37048b973ec9ac06bb15eeb26d370957736775
         e6c5638153f00152fe632215eb22f407dc3a1a2c473f45751b6451bf
         626b781d8db61c537cb431019fa8a6e74f3880adf74ce4760903d307
         940b6b5eacc590d7d51f03aafecc2ae9d93ca6763feb9d5545492546
         add30495d54deba8f0d6a9f583369e655c1ccbe6d6
```

B.7.4. Test vector for secret points K

```
       scalar_mult_vfy(ya,Yb): (length: 66 bytes)
         0057f1164b053e75197f7457d56391442efe0e5078fa9df658c6a6d0
         0af69ae3998d55f092ef9da4040bc42945691590496a5d84479439b1
         14d28b02a51ba41b273d
       scalar_mult_vfy(yb,Ya): (length: 66 bytes)
         0057f1164b053e75197f7457d56391442efe0e5078fa9df658c6a6d0
         0af69ae3998d55f092ef9da4040bc42945691590496a5d84479439b1
         14d28b02a51ba41b273d
```

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

```
transcript_ir(Ya,ADa,Yb,ADb): (length: 278 bytes)
  85010400493381a239f5c33edc50f92a4ac7dbade7854c2ef0905155
  67a5387c331fc9c7fdff4e151801a51ed20514412dd9832a0ebf4651
  e5d4cfebf1ebb89ca459a6311801d78c1142bf4eb46df664108bd05e
  8d979f3bc13ea3e373ce07fd67e457f44927e0123d7566b527eb10ea
  d21271d922f373fec78b4a3f4798f8fd52fb97859588de0341446185
  0104012938f18169cf9f37048b973ec9ac06bb15eeb26d3709577367
  75e6c5638153f00152fe632215eb22f407dc3a1a2c473f45751b6451
  bf626b781d8db61c537cb4310060575918b0c77f5208b31b89f6fc2c
  f86bf494a1533a6f282ae0fc550133d51626c35989c01462aabab6da
  b9522cfb6a2ab214570f29560a7cc9619aa3e334192903414462
DSI = G.DSI_ISK, b'CPaceP521_XMD:SHA-512_SSWU_NU__ISK':
(length: 34 bytes)
  435061636550353231 5f584d443a5348412d3531325f535357555f4e
  555f5f49534b
lv_cat(DSI,sid,K)||transcript_ir(Ya,ADa,Yb,ADb):
(length: 397 bytes)
  224350616365503532315f584d443a5348412d3531325f535357555f
  4e555f5f49534b107e4b4791d6a8ef019b936c79fb7f2c57420057f1
  164b053e75197f7457d56391442efe0e5078fa9df658c6a6d00af69a
  e3998d55f092ef9da4040bc42945691590496a5d84479439b114d28b
  02a51ba41b273d85010400493381a239f5c33edc50f92a4ac7dbade7
  854c2ef090515567a5387c331fc9c7fdff4e151801a51ed20514412d
  d9832a0ebf4651e5d4cfebf1ebb89ca459a6311801d78c1142bf4eb4
  6df664108bd05e8d979f3bc13ea3e373ce07fd67e457f44927e0123d
  7566b527eb10ead21271d922f373fec78b4a3f4798f8fd52fb978595
  88de0341446185010401 2938f18169cf9f37048b973ec9ac06bb15ee
  b26d370957736775e6c5638153f00152fe632215eb22f407dc3a1a2c
  473f45751b6451bf626b781d8db61c537cb4310060575918b0c77f52
  08b31b89f6fc2cf86bf494a1533a6f282ae0fc550133d51626c35989
  c01462aabab6dab9522cfb6a2ab214570f29560a7cc9619aa3e33419
  2903414462
ISK result: (length: 64 bytes)
  6f8daaedf8843dd9ae3b8376c6dd668f49a127e7d33b8adb37d4ae13
  5e1f09a68db1a03d436aae0cb8efb80a11813dec6924a52a6589b186
  e62e676a8dd777ad
```

B.7.6. Test vector for ISK calculation parallel execution

```
transcript_oc(Ya,ADa,Yb,ADb): (length: 280 bytes)
  6f63850104012938f18169cf9f37048b973ec9ac06bb15eeb26d3709
  57736775e6c5638153f00152fe632215eb22f407dc3a1a2c473f4575
  1b6451bf626b781d8db61c537cb4310060575918b0c77f5208b31b89
  f6fc2cf86bf494a1533a6f282ae0fc550133d51626c35989c01462aa
  bab6dab9522cfb6a2ab214570f29560a7cc9619aa3e3341929034144
  6285010400493381a239f5c33edc50f92a4ac7dbade7854c2ef09051
  5567a5387c331fc9c7fdff4e151801a51ed20514412dd9832a0ebf46
  51e5d4cfebf1ebb89ca459a6311801d78c1142bf4eb46df664108bd0
  5e8d979f3bc13ea3e373ce07fd67e457f44927e0123d7566b527eb10
  ead21271d922f373fec78b4a3f4798f8fd52fb97859588de03414461
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
  4e555f5f49534b107e4b4791d6a8ef019b936c79fb7f2c57420057f1
  164b053e75197f7457d56391442efe0e5078fa9df658c6a6d00af69a
  e3998d55f092ef9da4040bc42945691590496a5d84479439b114d28b
  02a51ba41b273d6f63850104012938f18169cf9f37048b973ec9ac06
  bb15eeb26d370957736775e6c5638153f00152fe632215eb22f407dc
  3a1a2c473f45751b6451bf626b781d8db61c537cb4310060575918b0
  c77f5208b31b89f6fc2cf86bf494a1533a6f282ae0fc550133d51626
  c35989c01462aabab6dab9522cfb6a2ab214570f29560a7cc9619aa3
  e3341929034144628501040049338 1a239f5c33edc50f92a4ac7dbad
  e7854c2ef090515567a5387c331fc9c7fdff4e151801a51ed2051441
  2dd9832a0ebf4651e5d4cfebf1ebb89ca459a6311801d78c1142bf4e
  b46df664108bd05e8d979f3bc13ea3e373ce07fd67e457f44927e012
  3d7566b527eb10ead21271d922f373fec78b4a3f4798f8fd52fb9785
  9588de03414461
ISK result: (length: 64 bytes)
  28c311ecace3f42a9cc2ec3cc7b8bcbfd5308d999bb795ae0af7a312
  9b83f36ef011d7186422ed03bf79725cdb1b17d2a995f6db753db6eb
  9438db72bdf23d77
```

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

```
      H.hash(b"CPaceSidOutput" + transcript_ir(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        eebbefc008cede41adf4cfb06c5f3d6236aa532a958dd352d4154efa
        bda569ae3afab954380b842cde2c13b4a9750d821634ce763be4c72d
        710fcc8d9f9c79f0
      H.hash(b"CPaceSidOutput" + transcript_oc(Ya,ADa, Yb,ADb)):
      (length: 64 bytes)
        3fc7eed0ea9381c98f60bc762c0bc6c73f90bb8fdc85c9826697838b
        76d116f0ee50a5489237b8bc7f30ab32157af2e8d8e6f712f07a3020
        eb112d609061f4f9
```

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[] = {
    0x0b,0x41,0x5f,0x69,0x6e,0x69,0x74,0x69,0x61,0x74,0x6f,0x72,
    0x0b,0x42,0x5f,0x72,0x65,0x73,0x70,0x6f,0x6e,0x64,0x65,0x72,
   };
   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[] = {
    0x04,0x00,0xae,0xfb,0x79,0xf4,0xba,0x40,0x36,0x9b,0xbe,0x78,
    0xa5,0xc8,0x73,0xb2,0xce,0x75,0xbe,0x11,0x15,0x16,0xde,0xed,
    0x28,0xc5,0x89,0x9a,0x8b,0xbf,0x8a,0x96,0xdb,0xd2,0x88,0x82,
    0xaf,0xbc,0x15,0x01,0xd8,0xa4,0xe9,0xb2,0x5c,0x17,0x2d,0xfa,
    0xdc,0xd9,0x6d,0x10,0x46,0x6f,0x9c,0x36,0xd2,0x3e,0xa0,0x5c,
    0x98,0x2d,0xf0,0xbb,0x1e,0x35,0x32,0x01,0xc9,0x8d,0xa6,0xce,
    0x14,0xe3,0xc4,0x24,0xb6,0xcf,0x55,0xe6,0x27,0xc4,0xf3,0xc0,
    0xa4,0x46,0x9e,0x2a,0xed,0xaa,0x69,0x2f,0x5f,0xe1,0x6d,0x57,
    0xba,0xa8,0xa9,0x43,0x2a,0xac,0x32,0x01,0x39,0xe8,0x54,0xd0,
    0x6b,0xfb,0x4d,0xaa,0xef,0xc5,0x04,0xff,0x2f,0xe1,0xd2,0xc4,
    0xf0,0x64,0x43,0x87,0x8a,0x24,0x2a,0x7b,0x87,0x6b,0x22,0x40,
    0x40,
   };
   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[] = {
 0x04,0x00,0x49,0x33,0x81,0xa2,0x39,0xf5,0xc3,0x3e,0xdc,0x50,
 0xf9,0x2a,0x4a,0xc7,0xdb,0xad,0xe7,0x85,0x4c,0x2e,0xf0,0x90,
 0x51,0x55,0x67,0xa5,0x38,0x7c,0x33,0x1f,0xc9,0xc7,0xfd,0xff,
 0x4e,0x15,0x18,0x01,0xa5,0x1e,0xd2,0x05,0x14,0x41,0x2d,0xd9,
 0x83,0x2a,0x0e,0xbf,0x46,0x51,0xe5,0xd4,0xcf,0xeb,0xf1,0xeb,
 0xb8,0x9c,0xa4,0x59,0xa6,0x31,0x18,0x01,0xd7,0x8c,0x11,0x42,
 0xbf,0x4e,0xb4,0x6d,0xf6,0x64,0x10,0x8b,0xd0,0x5e,0x8d,0x97,
 0x9f,0x3b,0xc1,0x3e,0xa3,0xe3,0x73,0xce,0x07,0xfd,0x67,0xe4,
 0x57,0xf4,0x49,0x27,0xe0,0x12,0x3d,0x75,0x66,0xb5,0x27,0xeb,
 0x10,0xea,0xd2,0x12,0x71,0xd9,0x22,0xf3,0x73,0xfe,0xc7,0x8b,
 0x4a,0x3f,0x47,0x98,0xf8,0xfd,0x52,0xfb,0x97,0x85,0x95,0x88,
 0xde,
};
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[] = {
 0x04,0x01,0x29,0x38,0xf1,0x81,0x69,0xcf,0x9f,0x37,0x04,0x8b,
 0x97,0x3e,0xc9,0xac,0x06,0xbb,0x15,0xee,0xb2,0x6d,0x37,0x09,
 0x57,0x73,0x67,0x75,0xe6,0xc5,0x63,0x81,0x53,0xf0,0x01,0x52,
 0xfe,0x63,0x22,0x15,0xeb,0x22,0xf4,0x07,0xdc,0x3a,0x1a,0x2c,
 0x47,0x3f,0x45,0x75,0x1b,0x64,0x51,0xbf,0x62,0x6b,0x78,0x1d,
 0x8d,0xb6,0x1c,0x53,0x7c,0xb4,0x31,0x00,0x60,0x57,0x59,0x18,
 0xb0,0xc7,0x7f,0x52,0x08,0xb3,0x1b,0x89,0xf6,0xfc,0x2c,0xf8,
 0x6b,0xf4,0x94,0xa1,0x53,0x3a,0x6f,0x28,0x2a,0xe0,0xfc,0x55,
 0x01,0x33,0xd5,0x16,0x26,0xc3,0x59,0x89,0xc0,0x14,0x62,0xaa,
 0xba,0xb6,0xda,0xb9,0x52,0x2c,0xfb,0x6a,0x2a,0xb2,0x14,0x57,
 0x0f,0x29,0x56,0x0a,0x7c,0xc9,0x61,0x9a,0xa3,0xe3,0x34,0x19,
 0x29,
};
const unsigned char tc_K[] = {
 0x00,0x57,0xf1,0x16,0x4b,0x05,0x3e,0x75,0x19,0x7f,0x74,0x57,
 0xd5,0x63,0x91,0x44,0x2e,0xfe,0x0e,0x50,0x78,0xfa,0x9d,0xf6,
 0x58,0xc6,0xa6,0xd0,0x0a,0xf6,0x9a,0xe3,0x99,0x8d,0x55,0xf0,
 0x92,0xef,0x9d,0xa4,0x04,0x0b,0xc4,0x29,0x45,0x69,0x15,0x90,
 0x49,0x6a,0x5d,0x84,0x47,0x94,0x39,0xb1,0x14,0xd2,0x8b,0x02,
 0xa5,0x1b,0xa4,0x1b,0x27,0x3d,
};
```

```
   const unsigned char tc_ISK_IR[] = {
    0x6f,0x8d,0xaa,0xed,0xf8,0x84,0x3d,0xd9,0xae,0x3b,0x83,0x76,
    0xc6,0xdd,0x66,0x8f,0x49,0xa1,0x27,0xe7,0xd3,0x3b,0x8a,0xdb,
    0x37,0xd4,0xae,0x13,0x5e,0x1f,0x09,0xa6,0x8d,0xb1,0xa0,0x3d,
    0x43,0x6a,0xae,0x0c,0xb8,0xef,0xb8,0x0a,0x11,0x81,0x3d,0xec,
    0x69,0x24,0xa5,0x2a,0x65,0x89,0xb1,0x86,0xe6,0x2e,0x67,0x6a,
    0x8d,0xd7,0x77,0xad,
   };
   const unsigned char tc_ISK_SY[] = {
    0x28,0xc3,0x11,0xec,0xac,0xe3,0xf4,0x2a,0x9c,0xc2,0xec,0x3c,
    0xc7,0xb8,0xbc,0xbf,0xd5,0x30,0x8d,0x99,0x9b,0xb7,0x95,0xae,
    0x0a,0xf7,0xa3,0x12,0x9b,0x83,0xf3,0x6e,0xf0,0x11,0xd7,0x18,
    0x64,0x22,0xed,0x03,0xbf,0x79,0x72,0x5c,0xdb,0x1b,0x17,0xd2,
    0xa9,0x95,0xf6,0xdb,0x75,0x3d,0xb6,0xeb,0x94,0x38,0xdb,0x72,
    0xbd,0xf2,0x3d,0x77,
   };
   const unsigned char tc_sid_out_ir[] = {
    0xee,0xbb,0xef,0xc0,0x08,0xce,0xde,0x41,0xad,0xf4,0xcf,0xb0,
    0x6c,0x5f,0x3d,0x62,0x36,0xaa,0x53,0x2a,0x95,0x8d,0xd3,0x52,
    0xd4,0x15,0x4e,0xfa,0xbd,0xa5,0x69,0xae,0x3a,0xfa,0xb9,0x54,
    0x38,0x0b,0x84,0x2c,0xde,0x2c,0x13,0xb4,0xa9,0x75,0x0d,0x82,
    0x16,0x34,0xce,0x76,0x3b,0xe4,0xc7,0x2d,0x71,0x0f,0xcc,0x8d,
    0x9f,0x9c,0x79,0xf0,
   };
   const unsigned char tc_sid_out_oc[] = {
    0x3f,0xc7,0xee,0xd0,0xea,0x93,0x81,0xc9,0x8f,0x60,0xbc,0x76,
    0x2c,0x0b,0xc6,0xc7,0x3f,0x90,0xbb,0x8f,0xdc,0x85,0xc9,0x82,
    0x66,0x97,0x83,0x8b,0x76,0xd1,0x16,0xf0,0xee,0x50,0xa5,0x48,
    0x92,0x37,0xb8,0xbc,0x7f,0x30,0xab,0x32,0x15,0x7a,0xf2,0xe8,
    0xd8,0xe6,0xf7,0x12,0xf0,0x7a,0x30,0x20,0xeb,0x11,0x2d,0x60,
    0x90,0x61,0xf4,0xf9,
   };
```

B.7.9. Testvectors as JSON file encoded as BASE64

```
    #eyJQUlMiOiAiNTA2MTczNzM3NzZGNzI2NCIsICJDSSI6ICIwQjQxNUY2OTZFNjk3ND
    #Y5NjE3NDZGNzIwQjQyNUY3MjY1NzM3MDZGNkU2NDY1NzIiLCAic2lkIjogIjdFNEI0
    #NzkxRDZBOEVGMDE5QjkzNkM3OUZCN0YyQzU3IiwgImciOiAiMDQwMEFFRkI3OUY0Qk
    #E0MDM2OUJCRTc4QTVDODczQjJDRTc1QkUxMTE1MTZERUVEMjhDNTg5OUE4QkJGOEE5
    #NkRCRDI4ODgyQUZCQzE1MDFEOEE0RTlCMjVDMTcyREZBRENEOTZEMTA0NjZGOUMzNk
    #QyM0VBMDVDOTgyREYwQkIxRTM1MzIwMUM5OERBNkNFMTRFM0M0MjRCNkNGNTVFNjI3
    #QzRGM0MwQTQ0NjlFMkFFREFBNjkyRjVGRTE2RDU3QkFBOEE5NDMyQUFDMzIwMTM5RT
    #g1NEQwNkJGQjREQUFFRkM1MDRGRjJGRTFEMkM0RjA2NDQzODc4QTI0MkE3Qjg3NkIy
    #MjQwNDAiLCAieWEiOiAiMDA2MzY3RTlDMkFFRkY5RjFEQjE5QUY2MDBDQ0E3MzM0M0
    #Q0N0NCRTQ0NkNFQkJEMUNDRDc4M0Y4Mjc1NUE4NzJEQTg2RkQwNzA3RUIzNzY3QzYx
    #MTRGMTgwM0RFQjYyRDYzQkREMUU2MTNGNjdFNjNFOEMxNDFFRTUzMTBFM0VFODE5Ii
    #wgIkFEYSI6ICI0MTQ0NjEiLCAiWWEiOiAiMDQwMDQ5MzM4MUEyMzlGNUMzM0VEQzUw
    #RjkyQTRBQzdEQkFERTc4NTRDMkVGMDkwNTE1NTY3QTUzODdDMzMxRkM5QzdGREZGNE
    #UxNTE4MDFBNTFFRDIwNTE0NDEyREQ5ODMyQTBFQkY0NjUxRTVENENGRUJGMUVCQjg5
    #Q0E0NTlBNjMxMTgwMUQ3OEMxMTQyQkY0RUI0NkRGNjY0MTA4QkQwNUU4RDk3OUYzQk
    #MxM0VBM0UzNzNDRTA3RkQ2N0U0NTdGNDQ5MjdFMDEyM0Q3NTY2QjUyN0VCMTBFQUQy
    #MTI3MUQ5MjJGMzczRkVDNzhCNEEzRjQ3OThGOEZENTJGQjk3ODU5NTg4REUiLCAieW
    #IiOiAiMDA5MjI3QkY4REM3NDFEQUNDOTQyMkY4QkYzQzBFOTZGQ0U5NTg3QkM1NjJF
    #QUFGRTBEQzVGNkY4MkYyODU5NEU0QTZGOTg1NTM1NjBDNjJCNzVGQTRBQkIxOThDRU
    #NCQkI4NkVCRDQxQjBFQTAyNTRDREU3OEFDNjhEMzlBMjQwQUU3IiwgIkFEYiI6ICI0
    #MTQ0NjIiLCAiWWIiOiAiMDQwMTI5MzhGMTgxNjlDRjlGMzcwNDhCOTczRUM5QUMwNk
    #JCMTVFRUIyNkQzNzA5NTc3MzY3NzVFNkM1NjM4MTUzRjAwMTUyRkU2MzIyMTVFQjIy
    #RjQwN0RDM0ExQTJDNDczRjQ1NzUxQjY0NTFCRjYyNkI3ODFEOERCNjFDNTM3Q0I0Mz
    #EwMDYwNTc1OTE4QjBDNzdGNTIwOEIzMUI4OUY2RkMyQ0Y4NkJGNDk0QTE1MzNBNkYy
    #ODJBRTBGQzU1MDEzM0Q1MTYyNkMzNTk4OUMwMTQ2MkFBQkFCNkRBQjk1MjJDRkI2QT
    #JBQjIxNDU3MEYyOTU2MEE3Q0M5NjE5QUEzRTMzNDE5MjkiLCAiSyI6ICIwMDU3RjEx
    #NjRCMDUzRTc1MTk3Rjc0NTdENTYzOTE0NDJFRkUwRTUwNzhGQTlERjY1OEM2QTZEMD
    #BBRjY5QUUzOTk4RDU1RjA5MkVGOURBNDA0MEJDNDI5NDU2OTE1OTA0OTZBNUQ4NDQ3
    #OTQzOUIxMTREMjhCMDJBNTFCQTQxQjI3M0QiLCAiSVNLX0lSIjogIjZGOERBQUVERj
    #g4NDNERDlBRTNCODM3NkM2REQ2NjhGNDlBMTI3RTdEMzNCOEFEQjM3RDRBRTEzNUUx
    #RjA5QTY4REIxQTAzRDQzNkFBRTBDQjhFRkI4MEExMTgxM0RFQzY5MjRBNTJBNjU4OU
    #IxODZFNjJFNjc2QThERDc3N0FEIiwgIklTS19TWSI6ICIyOEMzMTFFQ0FDRTNGNDJB
    #OUNDMkVDM0NDN0I4QkNCRkQ1MzA4RDk5OUJCNzk1QUUwQUY3QTMxMjlCODNGMzZFRj
    #AxMUQ3MTg2NDIyRUQwM0JGNzk3MjVDREIxQjE3RDJBOTk1RjZEQjc1M0RCNkVCOTQz
    #OERCNzJCREYyM0Q3NyIsICJzaWRfb3V0cHV0X2lyIjogIkVFQkJFRkMwMDhDRURFND
    #FBREY0Q0ZCMDZDNUYzRDYyMzZBQTUzMkE5NThERDM1MkQ0MTU0RUZBQkRBNTY5QUUz
    #QUZBQjk1NDM4MEI4NDJDREUyQzEzQjRBOTc1MEQ4MjE2MzRDRTc2M0JFNEM3MkQ3MT
    #BGQ0M4RDlGOUM3OUYwIiwgInNpZF9vdXRwdXRfb2MiOiAiM0ZDN0VFRDBFQTkzODFD
    #OThGNjBCQzc2MkMwQkM2QzczRjkwQkI4RkRDODVDOTgyNjY5NzgzOEI3NkQxMTZGME
    #VFNTBBNTQ4OTIzN0I4QkM3RjMwQUIzMjE1N0FGMkU4RDhFNkY3MTJGMDdBMzAyMEVC
    #MTEyRDYwOTA2MUY0RjkifQ==
```

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