lwig
Internet-Draft
Intended status: Informational
Expires: July 25, 2022

R. Struik
Struik Security Consultancy

Jan 21, 2022

Alternative Elliptic Curve Representations
draft-ietf-lwig-curve-representations-23

Abstract

This document specifies how to represent Montgomery curves and (twisted) Edwards curves as curves in short-Weierstrass form and illustrates how this can be used to carry out elliptic curve computations leveraging existing implementations and specifications of, e.g., ECDSA and ECDH using NIST prime curves. We also provide extensive background material that may be useful for implementers of elliptic curve cryptography.

Requirements Language

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.

Status of This Memo

Table of Contents

1. Fostering Code Reuse with New Elliptic Curves

Elliptic curves can be represented using different curve models. Recently, IETF standardized elliptic curves that are claimed to have better performance and improved robustness against "real world" attacks than curves represented in the traditional short-Weierstrass curve model. These so-called CFRG curves [RFC7748] use the Montgomery curve model and the model of twisted Edwards curves.

In this document, we specify these curves using the traditional short-Weierstrass model and also define how to efficiently switch between representations in these different curve models. In particular, we specify Wei25519, which allows an alternative representation of points of Curve25519 (a Montgomery curve) and of

points of Edwards25519 (a twisted Edwards curve), as points of a corresponding short-Weierstrass curve. Similarly, we specify Wei448, which allows an alternative representation of points of Curve448 (a Montgomery curve) and of points of Ed448 (an Edwards curve), as points of a corresponding short-Weierstrass curve.

Use of Wei25519 and Wei448 allows easy definition of new instantiations of signature schemes and key agreement schemes already specified for traditional NIST prime curves, thereby allowing easy integration with existing specifications, such as NIST SP 800-56a [SP-800-56a], FIPS Pub 186-4 [FIPS-186-4], and ANSI X9.62-2005 [ANSI-X9.62], and fostering code reuse on platforms that already implement some of these schemes using elliptic curve arithmetic for curves in short-Weierstrass form (see Appendix C.1). To illustrate this, we specify how to use Wei25519 and Wei448 with co-factor ECDH and with ECDSA, thereby giving rise to the key agreement schemes ECDH25519 and ECDH448 and the signature schemes ECDSA25519 and ECDSA448. In all these cases, implementors may use the curve arithmetic for the curve model of their choosing (where they can efficiently switch between representations in different curve models, if required).

For ease of exposition, we consider Wei25519 first and introduce Wei448 simply as an illustration of how to create other "offspring" objects and protocols (see Section 4.4). We also provide extensive background material that we hope may be useful for implementors of elliptic curve cryptography or for cross-referencing with future specification work.

2. Specification of Wei25519

For the specification of Wei25519 and its relationship to Curve25519 and Edwards25519, see Appendix E. For further details and background information on elliptic curves, we refer to the other appendices.

The use of Wei25519 allows reuse of existing generic code that implements short-Weierstrass curves, such as the NIST curve P-256, to also implement the CFRG curves Curve25519 and Edwards25519. (Here, generic code refers to an implementation that does not depend on hardcoded domain parameters (see also Section 6).) We also cater to reusing of existing code where some domain parameters may have been hardcoded, thereby widening the scope of applicability. To this end, we specify the short-Weierstrass curves Wei25519.2 and Wei25519.-3, with hardcoded domain parameter a=2 and a=-3 (mod p), respectively; see Appendix G. (Here, p is the characteristic of the field over which these curves are defined.)

3. Use of Representation Switches

The curves Curve25519, Edwards25519, and Wei25519, as specified in Appendix E.3, are all isomorphic, with the transformations of Appendix E.2. These transformations map the specified base point of each of these curves to the specified base point of each of the other curves. Consequently, a public-private key pair (k,R:=k*G) for any one of these curves corresponds, via these isomorphic mappings, to the public-private key pair (k, R':=k*G') for each of these other curves (where G and G' are the corresponding base points of these curves). This observation extends to the case where one also considers curve Wei25519.2 (which has hardcoded domain parameter a=2), as specified in Appendix G.3, since it is isomorphic to Wei25519, with the transformation of Appendix G.2, and, thereby, also isomorphic to Curve25519 and Edwards25519.

The curve Wei25519.-3 (which has hardcoded domain parameter a=-3 (mod p)) is not isomorphic to the curve Wei25519, but is related in a slightly weaker sense: the curve Wei25519 is isogenous to the curve Wei25519.-3, where the mapping of Appendix G.2 is an isogeny of degree l=47 that maps the specified base point G of Wei25519 to the specified base point G' of Wei25519.-3 and where the so-called dual isogeny (which maps Wei25519.-3 to Wei25519) has the same degree l=47, but does not map G' to G, but to a fixed multiple hereof, where this multiple is l=47. Consequently, a public-private key pair (k,R:=k*G) for Wei25519 corresponds to the public-private key pair (k, R':= k*G') for Wei25519.-3 (via the l-isogeny), whereas the public-private key pair (k, R':=k*G') corresponds to the public-private key pair (l*k, l*R=l*k*G) of Wei25519 (via the dual isogeny). (Note the extra scalar l=47 here.)

Alternative curve representations can, therefore, be used in any cryptographic scheme that involves computations on public-private key pairs, where implementations may carry out computations on the corresponding object for the isomorphic or isogenous curve and convert the results back to the original curve (where, in case this involves an l-isogeny, one has to take into account the factor l). This includes use with elliptic-curve based signature schemes and key agreement and key transport schemes.

For some examples of curve computations on each of the curves specified in Appendix E.3 and Appendix G.3, see Appendix J.

4. Examples

4.1. Implementation of X25519, Specification of ECDH25519

RFC 7748 [RFC7748] specifies the use of X25519, a co-factor Diffie-Hellman key agreement scheme, with instantiation by the Montgomery curve Curve25519. This key agreement scheme was already specified in Section 6.1.2.2 of NIST SP 800-56a [SP-800-56a] for elliptic curves in short-Weierstrass form. Hence, one can implement X25519 using existing NIST routines by (1) representing a point of the Montgomery curve Curve25519 as a point of the Weierstrass curve Wei25519; (2) instantiating the co-factor Diffie-Hellman key agreement scheme of the NIST specification with the resulting point and Wei25519 domain parameters; (3) representing the key resulting from this scheme (which is a point of the curve Wei25519 in Weierstrass form) as a point of the Montgomery curve Curve25519. The representation change can be implemented via a simple wrapper and involves a single modular addition (see Appendix E.2). Using this method has the additional advantage that one can reuse the public-private key pair routines, domain parameter validation, and other checks that are already part of the NIST specifications.

A NIST-compliant version of the co-factor Diffie-Hellman key agreement scheme (denoted by ECDH25519) results if one keeps inputs (key contributions) and pre-output (shared key K) in the short-Weierstrass format (and, hence, does not perform Steps (1) and (3) above), where the actual output (shared secret Z) is the x-coordinate of K (if this is an affine point of the curve), represented as a fixed-size octet string in tight MSB/msb-order using the FE2OS mapping of Appendix I.5, and where the output is an error indicator otherwise (i.e., if K is the point at infinity O of the curve).

NOTE 1: A Montgomery version of the co-factor Diffie-Hellman key agreement scheme (denoted by X25519+) results by incorporating Steps (1), (2), and (3) above, i.e., where one keeps inputs (key contributions) and pre-output (shared key K) in the Montgomery curve format, as points of Curve25519, where one represents each affine point by only its x-coordinate, represented as a fixed-size octet string in tight LSB/msb-order using the FE2OS mapping and its reverse, the strict OS2FE mapping, of Appendix I.5, and where the actual output (shared secret Z) is the representation of the shared key K as defined above (if this is an affine point of the curve), and where the output is an error indicator otherwise (i.e., if K is the point at infinity O of the curve). The scheme X25519, as specified in [RFC7748], is a more lenient version of this X25519+ scheme, whereby one does not mandate rejection of shared keys in the small subgroup (which are instead represented as if these were the point (0,0) of order two), where one does not check whether a received key contribution is a point of Curve25519 rather than a point of a quadratic twist of this curve (for definitions of these terms, see

Appendix B.1), and where one uses the non-strict (rather than strict) OS2FE mapping (which, in this case, is always applied after setting the leftmost bit of the rightmost octet to zero). Moreover, with X25519, private keys are generated in the interval [2^251,2^252-1] rather than in the interval [1,n-1] (the so-called "clamping") and one uses as base point G':=h*G, where G, n, and h are, respectively, the fixed base point, the order of the base point, and the co-factor of the curve in question.

NOTE 2: At this point, it is unclear whether a FIPS-accredited module implementing the co-factor Diffie-Hellman scheme with, e.g., P-256 would also extend this accreditation to the Montgomery versions X25519+ or X25519. (For cryptographic module validation program guidance, see, e.g., [FIPS-140-2].)

4.2. Implementation of Ed25519

RFC 8032 [RFC8032] specifies Ed25519, a "full" Schnorr signature scheme, with instantiation by the twisted Edwards curve Edwards25519. One can implement the computation of the ephemeral key pair for Ed25519 using an existing Montgomery curve implementation by (1) generating a random public-private key pair (k, R':=k*G') for Curve25519; (2) representing this public-private key as the pair (k, R:=k*G) for Ed25519. As before, the representation change can be implemented via a simple wrapper. Note that the Montgomery ladder specified in Section 5 of RFC7748 [RFC7748] does not provide sufficient information to reconstruct R':=(u, v) (since it does not compute the v-coordinate of R'). However, this deficiency can be remedied by using a slightly modified version of the Montgomery ladder that includes reconstruction of the v-coordinate of R':=k*G' at the end of the Montgomery ladder (which uses the v-coordinate of the base point G' of Curve25519 as well). For details, see Appendix C.2.

4.3. Specification of ECDSA25519

FIPS Pub 186-4 [FIPS-186-4] specifies the signature scheme ECDSA and can be instantiated not just with the NIST prime curves, but also with other Weierstrass curves (that satisfy additional cryptographic criteria). In particular, one can instantiate this scheme with the Weierstrass curve Wei25519 and the hash function SHA-256 [FIPS-180-4], where an implementation may generate an ephemeral public-private key pair for Wei25519 by (1) internally carrying out these computations on the Montgomery curve Curve25519, the twisted Edwards curve Edwards25519, or even the Weierstrass curve Wei25519.-3 (with hardcoded a=-3 domain parameter); (2) representing the result as a key pair for the curve Wei25519. Note that, in either case, one can implement these schemes with the same representation conventions

as used with existing NIST specifications, including bit/byte-ordering, compression functions, and the like. This allows generic implementations of ECDSA with the hash function SHA-256 and with the NIST curve P-256 or with the curve Wei25519 specified in this specification to reuse the same implementation (instantiated with, respectively, the NIST P-256 elliptic curve domain parameters or with the domain parameters of curve Wei25519 specified in Appendix E). We denote by ECDSA25519 the instantiation of ECDSA with SHA-256 and with curve Wei25519, where the signature (r,s) is represented as the right-concatenation of the integers r and s in the interval [1,n-1], where n is the order of the base point of the curve in question, each represented as fixed-size octet strings in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6.

4.4. Other Uses (Wei448, ECDH448, ECDSA448, and Others)

Any existing specification of cryptographic schemes using elliptic curves in Weierstrass form and that allows introduction of a new elliptic curve (here: Wei25519) is amenable to similar constructs, thus spawning "offspring" protocols, simply by instantiating these using the new curve in short-Weierstrass form, thereby allowing code and/or specifications reuse and, for implementations that so desire, carrying out curve computations "under the hood" on Montgomery curve and twisted Edwards curve cousins hereof (where these exist). This would simply require definition of a new object identifier for any such envisioned "offspring" protocol. This could significantly simplify standardization of schemes and help keeping at bay the resource and maintenance cost of implementations supporting algorithm agility [RFC7696].

We illustrate the construction of such offspring protocols for Curve448, another Montgomery curve recently standardized by IETF (see [RFC7748]). Similar to the case with Curve25519, one can represent points of this curve via different curve models, viz. as points of an Edwards curve (Ed448) or as points of a short-Weierstrass curve (Wei448). For the specification of Wei448 and its relationship to Curve448 and Ed448, see Appendix M. As with ECDH25519, one can now easily define a NIST-compliant version of co-factor Diffie-Hellman key agreement (denoted by ECDH448), by simply reusing the example of Section 4.1, but now using the short-Weierstrass curve Wei448, rather than Wei25519 (with the same representation and bit/byte-ordering conventions). Similarly, one can easily specify ECDSA with Wei448 and a suitable hash function, by simply reusing the example of Section 4.3, but now using the short-Weierstrass curve Wei448, rather than Wei25519, and picking as hash function SHAKE256 (see Section 6.3 of [FIPS-202]) with output size of d0=512 bits. We denote by ECDSA448 the resulting signature scheme (with the same representation and bit/byte-ordering conventions).

NOTE: A Montgomery version of the co-factor Diffie-Hellman key agreement scheme (denoted by X448+) results by reusing the description of X25519+ in Section 4.1, but now using the Montgomery curve Curve448, rather than Curve25519 (with the same checks and representation and bit/byte-ordering conventions). The scheme X448, as specified in [RFC7748], is a more lenient version of this X448+ scheme, whereby one does not mandate rejection of shared keys in the small subgroup (which are instead represented as if these were the point (0,0) of order two), nor checks whether a received key contribution is a point of Curve448 rather than a point of a quadratic twist of this curve, and where one uses the non-strict (rather than the strict) OS2FE mapping for converting octet strings to field elements. Moreover, with X448, private keys are generated in the interval [2^445,2^446-1] rather than in the interval [1,n-1] (the so-called "clamping") and one uses as base point G':=h*G, where G, n, and h are, respectively, the fixed base point, the order of the base point, and the co-factor of the curve in question.

5. Caveats

The examples above illustrate how specifying the Weierstrass curve Wei25519 (or any curve in short-Weierstrass format, for that matter) may facilitate reuse of existing code and may simplify standards development. However, the following caveats apply:

5.1. Wire Format

The transformations between alternative curve representations can be implemented at negligible relative incremental cost if the curve points are represented as affine points. If a point is represented in compressed format, conversion usually requires a costly point decompression step. This is the case in [RFC7748], where the inputs to the co-factor Diffie-Hellman scheme X25519, as well as its output, are represented in u-coordinate-only format. This is also the case in [RFC8032], where the EdDSA signature includes the ephemeral signing key represented in compressed format (see Appendix H for details). Note that in the latter case compression is lossless, whereas it is lossy in the former case.

5.2. Representation Conventions

While elliptic curve computations are carried-out in a field GF(q) and, thereby, involve large integer arithmetic, these integers are represented as bit- and byte-strings. Here, [RFC8032] uses least-significant-byte (LSB)/least-significant-bit (lsb) conventions, whereas [RFC7748] uses LSB/most-significant-bit (msb) conventions, and where most other cryptographic specifications, including NIST SP800-56a [SP-800-56a], FIPS Pub 186-4 [FIPS-186-4], and ANSI

X9.62-2005 [ANSI-X9.62] use most-significant-byte (MSB)/msb conventions. Since each pair of conventions is different (see Appendix I for details and Appendix J for examples), this does necessitate bit/byte representation conversions.

5.3. Domain Parameters

All traditional NIST curves are Weierstrass curves with domain parameter a=-3, while all Brainpool curves [RFC5639] are isomorphic to a Weierstrass curve of this form. Thus, one can expect there to be existing Weierstrass implementations with a hardcoded a=-3 domain parameter ("Jacobian-friendly"). For those implementations, including the curve Wei25519 as a potential vehicle for offering support for the CFRG curves Curve25519 and Edwards25519 is not possible, since it is not of the required form. Instead, one has to implement Wei25519.-3 and include code that implements the isogeny and dual isogeny from and to Wei25519. The lowest odd-degree isogeny has degree l=47 and requires roughly 9kB of storage for isogeny and dual-isogeny computations (see the tables in Appendix G.4). Note that storage would have reduced to a single 64-byte table if only the Curve25519 curve would have been generated so as to be isomorphic to a Weierstrass curve with hardcoded a=-3 parameter (this corresponds to l=1).

NOTE 1: An example of a Montgomery curve defined over the same field as Curve25519 that is isomorphic to a Weierstrass curve with hardcoded a=-3 parameter is the Montgomery curve M_{A,B} with B=1 and A=-1410290 (or, if one wants the base point to still have u-coordinate u=9, with B=1 and A=-3960846). In either case, the resulting curve has the same cryptographic properties as Curve25519 and the same performance (which relies on A being a 3-byte integer, as is the case with the domain parameter A=486662 of Curve25519, and using the same special prime p=2^255-19), while at the same time being "Jacobian-friendly" by design.

NOTE 2: While an implementation of Curve25519 via an isogenous Weierstrass curve with domain parameter a=-3 requires a relatively large table (of size roughly 9kB), for a quadratic twist of Curve25519 (e.g., the Montgomery curve M_{A,B'} with A=486662 and B'=2) this implementation approach only requires a table of size less than 0.5kB (over 20x smaller), solely due to the fact that it is l-isogenous to a Weierstrass curve with a=-3 parameter with relatively small parameter l=2 (compared to l=47, as is the case with Curve25519 itself).

6. Implementation Considerations

The efficiency of elliptic curve arithmetic is primarily determined by the efficiency of its group operations (see Appendix C). Numerous optimized formulae exist, such as the use of so-called Montgomery ladders with Montgomery curves [Mont-Ladder] or with Weierstrass curves [Wei-Ladder], the use of hardcoded a=-3 domain parameter for Weierstrass curves [ECC-Isogeny], and the use of hardcoded a=-1 domain parameters for twisted Edwards curves [tEd-Formulas]. These all target reduction of the number of finite field operations (primarily, finite field multiplications and squarings). Other optimizations target more efficient modular reductions underlying these finite field operations, by specifying curves defined over a field GF(q), where the field size q has a special form or a specific bit-length (typically, close to a multiple of a machine word). Depending on the implementation strategy, the bit-length of q may also facilitate reduced so-called "carry-effects" of integer arithmetic.

Most curves use a combination of these design philosophies. All NIST curves [FIPS-186-4] and Brainpool curves [RFC5639] are Weierstrass curves with a=-3 domain parameter, thus facilitating more efficient elliptic curve group operations than with a<>-3 (via so-called Jacobian coordinates). The NIST curves and the Montgomery curve Curve25519 are defined over prime fields, where the prime number has a special form, whereas the Brainpool curves - by design - use a generic prime number. None of the NIST prime curves, nor the Brainpool curves, can be expressed as Montgomery or twisted Edwards curves, whereas - conversely - Montgomery curves and twisted curves can be expressed as Weierstrass curves.

While use of Wei25519 allows reuse of existing generic code that implements short-Weierstrass curves, such as the NIST curve P-256, to also implement the CFRG curves Curve25519 or Edwards25519, this obviously does not result in an implementation of these CFRG curves that exploits the specific structure of the underlying field or other specific domain parameters (since generic). Reuse of generic code, therefore, may result in a less computationally efficient curve implementation than would have been possible if the implementation had specifically targeted Curve25519 or Edwards25519 alone (with the overall cost differential estimated to be somewhere in the interval [1.00-1.25]). If existing generic code offers hardware support, however, the overall speed may still be larger, since less efficient formulae for curve arithmetic using Wei25519 curves compared to a direct implementation of Curve25519 or Edwards25519 arithmetic may be more than compensated for by faster implementations of the finite field arithmetic itself.

Overall, one should consider not just code reuse and computational efficiency, but also development and maintenance cost, and, e.g, the cost of providing effective implementation attack countermeasures (see also Section 8).

7. Implementation Status

[Note to the RFC Editor] Please remove this entire section before publication, as well as the reference to [RFC7942].

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit.

Nikolas Rosener evaluated the performance of switching between different curve models in his Master's thesis [Rosener]. For an implementation of Wei25519, see <https://github.com/ncme/c25519>. For support of this curve in tinydtls, see <https://github.com/ncme/tinydtls>.

ANSSI (the national cybersecurity agency of France) implemented the Ed25519 signature scheme using a generic ECC library for short-Weierstrass curves instantiated with the Wei25519 domain parameters, where this was motivated by the desire to both keep the library core mathematical foundations simple and keep the defense-in-depth (regarding software security and side-channels) focused on a rather limited part. For further details, see <https://githubmemory.com/index.php/repo/ANSSI-FR/libecc>.

According to <https://community.nxp.com/docs/DOC-330199>, an implementation of Wei25519 on the Kinets LTC ECC HW platform improves

the performance by over a factor ten compared to a stand-alone implementation of Curve25519 without hardware support.

The signature scheme ECDSA25519 (see Section 4.3) is supported in [RFC8928].

8. Security Considerations

The different representations of elliptic curve points discussed in this document are all obtained using a publicly known transformation, which is either an isomorphism or a low-degree isogeny. It is well-known that an isomorphism maps elliptic curve points to equivalent mathematical objects and that the complexity of cryptographic problems (such as the discrete logarithm problem) of curves related via a low-degree isogeny are tightly related. Thus, the use of these techniques does not negatively impact cryptographic security of elliptic curve operations.

As to implementation security, reusing existing high-quality code or generic implementations that have been carefully designed to withstand implementation attacks for one curve model may allow a more economical way of development and maintenance than providing this same functionality for each curve model separately (if multiple curve models need to be supported) and, otherwise, may allow a more gradual migration path, where one may initially use existing and accredited chipsets that cater to the pre-dominant curve model used in practice for over 15 years.

Elliptic curves are generally used as objects in a broader cryptographic scheme that may include processing steps that depend on the representation conventions used (such as with, e.g., key derivation following key establishment). These schemes should (obviously) unambiguously specify fixed representations of each input and output (e.g., representing each elliptic curve point always in short-Weierstrass form and in uncompressed tight MSB/msb format).

To prevent cross-protocol attacks, private keys SHOULD only be used with one cryptographic scheme.

Private keys MUST NOT be reused between Ed25519 (as specified in [RFC8032]) and ECDSA25519 (as specified in Section 4.3). Similarly, private keys MUST NOT be reused between Ed448 (as specified in [RFC8032]) and ECDSA448 (as specified in Section 4.4).

To prevent intra-protocol cross-instantiation attacks, ephemeral private keys MUST NOT be reused between instantiations of ECDSA25519 or of ECDSA448.

With ECDSA25519 and ECDSA448, the same private signature key MUST NOT be reused between application scenarios where message encoding and decoding rules vary, since this may jeopardize message unforgeability properties; see also the Note in Section 10.2.1. (In fact, this holds for any signature scheme, not just ECDSA.)

9. Privacy Considerations

The transformations between different curve models described in this document are publicly known and, therefore, do not affect privacy provisions.

Use of a public key in any protocol for which successful execution evidences knowledge of the corresponding private key implicitly indicates the entity holding this private key. Reuse of this public key with more than one protocol or more than one protocol instantiation may, therefore, allow traceability of this entity. It may also allow correlation of meta-data communicated with this common data element (e.g., different addressing information), even if an observer cannot technically verify the binding of this meta-data.

The randomized representation described in Appendix K.5 allows random curve points to be represented as random pairs of field elements, thereby assisting in obfuscating the presence of these curve points in some applications. For representations as random binary strings, see Appendix K.6.

10. Using Wei25519 and Wei448 with COSE and JOSE

This section defines algorithm encodings and representations enabling the use of the curves Wei25519 and Wei448 and their use with ECDH and ECDSA with JOSE [RFC7518] and COSE [RFC8152] messages.

All octet string encodings below use the MSB/msb-ordering conventions as defined in Appendix I.7. For CBOR representation details, we refer to [RFC8949]; for base64url encodings, we refer to [RFC4648].

10.1. Using Wei25519 and Wei448 Keys with COSE and JOSE

For Weierstrass curves, the representation of the point at infinity O is curve-specific (see Appendix H.1). For the short-Weierstrass curve Wei25519, we define O:=(-1,0), whereas for Wei448, we define O:=(1,0).

The encodings below specify the use of short-Weierstrass curves with COSE (see Section 10.1.1) and JOSE (see Section 10.1.2), where the encoding for a specific curve results by setting the "crv" parameter

to the unique name of the curve in question (i.e., "Wei25519" for the curve Wei25519 and "Wei448" for the curve Wei448).

10.1.1. Encoding of Short-Weierstrass Curves with COSE

With COSE, points of short-Weierstrass curves are encoded using the "EC2" key type (Section 13.1.1 of [RFC8152]) or the "OKP" key type (Section 7.2 of [I-D.ietf-cose-rfc8152bis-algs]), which are instantiated by setting the "crv" parameter to the (unique) name of the curve in question and the "kty" parameter to "EC2" or "OKP", respectively, where key type-specific settings are as follows:

a. With the "EC2" type, each affine point (X, Y) is encoded by setting the parameters "x" and "y" to the octet string representations of the elements X and Y, respectively, in tight MSB/msb-order, and converting each to a CBOR byte string. Each compressed point (X, t) is encoded by setting the parameter "x" to the octet representation of the element X, in tight MSB/msb-order, converted to a CBOR byte string, and by setting the parameter "y" to the CBOR false or CBOR true value, depending on whether, respectively, t=0 or t=1. For representation details and for details on the reverse mappings, see Appendix I.8. (Note that for affine points of a curve defined over a prime field this representation is consistent with the "EC2" representation in Section 13.1.1 of [RFC8152].)

b. With the "OKP" type, each point is encoded by setting the parameter "x" to the "squeezed" point representation of this point, in MSB/msb-order, and converting this to a CBOR byte string. For representation details and for details on the reverse mappings, see Appendix I.8. (Note that for affine points of a curve defined over a prime field this representation is consistent with the "OKP" representation in Section 7.2 of [I-D.ietf-cose-rfc8152bis-algs], which affords a curve-specific octet string encoding.)

In either case, if the point is a public key (i.e., the private key is well-defined), the parameter "d" encodes the corresponding private key, using the octet string representation, in tight MSB/msb-order, and converting this to a CBOR byte string (see Appendix I.6).

For curve points, the "crv" parameter and the parameters referenced with the applicable key type-specific settings above MUST be present in the structure, whereas the parameter "d" MUST NOT be present, while for private keys, the parameters "crv" and "d" MUST be present and the applicable key type-specific parameters of the corresponding public-key are RECOMMENDED to be present.

10.1.2. Encoding of Short-Weierstrass Curves with JOSE

With JOSE, points of short-Weierstrass curves are encoded using the "EC" key type (Section 6.2 of [RFC7518]) or the "OKP" key type (Section 2 of [RFC8037]), which are instantiated by setting the "crv" parameter to the (unique) name of the curve in question and the "kty" parameter to "EC" or "OKP", respectively, where key type-specific settings are as follows:

a. With the "EC" type, each affine curve point (X, Y) is encoded by setting the parameters "x" and "y" to the octet string representations of the elements X and Y, respectively, in tight MSB/msb-order, and converting each using the base64url encoding. The point at infinity O is encoded as if this were an affine point. For representation details and details on the reverse mappings, see Appendix I.8. (Note that for affine points of a curve defined over a prime field this representation is consistent with the "EC" representation in Section 6.2 of [RFC7518]).)

b. With the "OKP" type, each curve point is encoded by setting the parameter "x" to the "squeezed" point representation of this point, in MSB/msb-order, and converting this using the base64url encoding. For representation details and for details on the reverse mappings, see Appendix I.8. (Note that for affine points of a curve defined over a prime field this representation is consistent with the "OKP" representation in Section 2 of [RFC8037], which affords a curve-specific octet string encoding.)

In either case, if the point is a public key (i.e., the private key is well-defined), the parameter "d" encodes the corresponding private key, using the octet string representation, in tight MSB/msb-order, and converting this using the base64url encoding (see Appendix I.6).

For curve points, the "crv" parameter and the parameters referenced with the applicable key type-specific settings above MUST be present in the structure, whereas the parameter "d" MUST NOT be present, while for private keys, the parameters "crv" and "d" MUST be present and the applicable key type-specific parameters of the corresponding public-key are RECOMMENDED to be present.

10.2. Using ECDSA25519 and ECDSA448 with COSE and JOSE

FIPS Pub 186-4 [FIPS-186-4] specifies the signature scheme ECDSA and can be instantiated with suitable combinations of elliptic curves in short-Weierstrass form and hash functions (that satisfy particular cryptographic criteria). While this completely specifies the

internal workings of the signing and signature verification operations, this does not uniquely specify the input/output formats:

a. The signing operation takes as inputs a message m (represented as a bit string) and a private key d in the interval [1,n-1] and produces as output a signature, which is an ordered pair (r, s) of integers in the interval [1,n-1], where n is the order of the base point of the curve in question;

b. The signature verification operation takes as inputs a message m, a public key Q, and a signature (r,s) and produces as output the value "valid" or "invalid", depending upon whether the message was purportedly signed by a holder of the private key of the public-private key pair (d, Q) for the curve used with the signature scheme in question.

All inputs and outputs are uniquely determined by specifying the encodings of the message m, the private key d, the public key Q, the signature, and the values "valid" and "invalid".

The encodings below specify the use of instantiations of ECDSA with COSE (see Section 10.2.1) and JOSE (see Section 10.2.2), where the encoding for a specific ECDSA instantiation (i.e., with a specific short-Weierstrass curve and specific hash function) results by setting the "crv" parameter to the unique name of the underlying curve in question and the "alg" parameter to the unique name of the specific signature scheme instantiation. For JOSE, this is realized by setting the "alg" parameter to "ECDSA25519" for the ECDSA scheme defined in Section 4.3 and to "ECDSA448" for the scheme defined in Section 4.4. For COSE, this is realized by setting the "alg" parameter to "ES256" (short-hand for "ECDSA with SHA-256") for the ECDSA scheme defined in Section 4.3 and to "ECDSA with SHAKE256" for the scheme defined in Section 4.4. Note that, in the case of JOSE, the "alg" name uniquely defines the curve (and, thereby, implicitly the underlying "crv" parameter) and the underlying hash function, while in the case of COSE, the "alg" name uniquely defines the underlying hash function, but not the underlying curve.

10.2.1. Encoding of ECDSA Instantiations with COSE

Instantiations of ECDSA used with COSE use the following encodings of inputs and outputs:

a. The message m is the COSE Sig_structure as specified in Section 4.4 of [RFC8152], converted to the CBOR byte string ToBeSigned in accordance with the Core Deterministic Encoding Requirements of Section 4.2.1 of [RFC8949]), converted to a bit string using the OS2BS mapping of Appendix I.4;

b. The public key Q and the private key d are encoded as specified in Section 10.1.1, where the "crv" parameter is set to the unique name of the curve used with this particular instantiation of ECDSA;

c. The Cose signature is encoded as the right-concatenation of the octet string representations of the coordinates of the signature pair (r, s), in left-to-right order, where r and s are each represented as octet strings in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6, converted to a CBOR byte string. Note that, since we use a tight representation, this right-concatenated octet string has fixed size 2*l, where the parameter l is uniquely defined by the set Z_n in question (where n is the (prime) order of the base point of the curve in question). The inverse mapping results by checking that the purported encoded signature (after CBOR decoding) has indeed size 2*l, and by converting the left-side and right-side halves of this octet string (each of length l) to, respectively, the integers r and s in Z_n, via the strict OS2ZnE mapping of Appendix I.6.

When using a COSE key for this algorithm, if the "alg" field is present, it MUST be set to the (unique) name of this particular instantiation of ECDSA and the "crv" parameter MUST be set to the (unique) name of the corresponding curve; if the "key_ops" field is present, it MUST include "sign" when creating an ECDSA signature and it MUST include "verify" when verifying an ECDSA signature.

NOTE: Care should be taken that signers and verifiers do have a common understanding of message encoding rules, since otherwise signature verification may fail for messages with the same semantics. As an example, if there is ambiguity as to whether to represent the binary digit 0 as the integer 0 or as the CBOR false value (represented as the CBOR bit string b000_00000 or b111_10100, respectively), signing and signature verification may depend on different ToBeSigned strings and, thereby, may fail unexpectedly. This explains the (strong) requirement for deterministic encoding rules above and, thereby, the requirement for strong typing of any CBOR encodings used with signed messages. Further care should be taken that message decoding rules are always unambiguous, since otherwise the semantics of signed messages may not be clear or the unforgeability property of signatures may be jeopardized.

10.2.2. Encoding of ECDSA Instantiations with JOSE

Instantiations of ECDSA used with JOSE use the following encodings of inputs and outputs:

a. The message m is the JWS Signing Input as specified in [RFC7515], converted to a bit string, using the OS2BS mapping of Appendix I.4;

b. The public key and the private key are encoded as specified in Section 10.1.2, where the "crv" parameter is set to the unique name of the curve used with this particular instantiation of ECDSA;

c. The JWS signature is encoded as the right-concatenation of the octet string representations of the coordinates of the signature pair (r, s), in left-to-right order, where r and s are each represented as octet strings in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6, converted using the base64url encoding. Note that, since we use a tight representation, this right-concatenated octet string has fixed size 2*l, where the parameter l is uniquely defined by the set Z_n in question (where n is the (prime) order of the base point of the curve in question). The inverse mapping results by checking that the purported encoded signature (after base64url decoding) has indeed size 2*l, and by converting the left-side and right-side halves of this octet string (each of length l) to, respectively, the integers r and s in Z_n, via the strict OS2ZnE mapping of Appendix I.6.

When using a JOSE key for this algorithm, if the "alg" field is present, it MUST be set to the (unique) name of this particular instantiation of ECDSA and the "crv" parameter MUST be set to the (unique) name of the corresponding curve; if the "key_ops" field is present, it MUST include "sign" when creating an ECDSA signature and it MUST include "verify" when verifying an ECDSA signature; if the JWK _use_ field is present, its value MUST be "sig".

10.3. Using ECDH25519 and ECDH448 with COSE and JOSE

Section 6.1.2.2 of NIST SP 800-56a [SP-800-56a] specifies the co-factor elliptic-curve Diffie-Hellman key agreement scheme (co-factor ECDH) and can be instantiated with a suitable elliptic curve in short-Weierstrass form (that satisfies particular cryptographic criteria). While this completely specifies the internal workings of the key agreement scheme in question, this does not uniquely specify the input/output formats:

a. The co-factor Diffie-Hellman primitive (Section 5.7.1.2 of [SP-800-56a]) takes as inputs a private key d in the interval [1,n-1] from one of the parties and a point Q' obtained from the other party and produces the shared key K:=h*(d*Q'), where h and n are, respectively, the co-factor and the order of the base

point of the curve in question and where Q' is a point of this curve. If this shared key K is the point at infinity O of the curve, the output is an error indicator;

b. If the shared key K is an affine point of the curve, the output is the (raw) shared secret Z, which is the fixed-size octet representation of the x-coordinate of K, using the FE2OS mapping of Appendix I.5, represented in tight-MSB/msb-order (see Appendix I.7).

(NOTE: A subsequent key derivation function (kdf) takes as inputs the shared secret Z and side information OtherInfo and produces as output an octet string of DerivedKeyingMaterial, where details depend on the used kdf in question. This step is out of scope.)

The inputs and outputs are uniquely determined by specifying the encodings of private keys, curve points, and the error indicator for this key agreement scheme.

The encodings below specify the use of instantiations of ECDH with COSE (see Section 10.3.1) and JOSE (see Section 10.3.2), where the encoding for a specific co-factor ECDH instantiation (i.e., with a specific short-Weierstrass curve) results by setting the "crv" parameter to the unique name of the underlying curve in question and the "alg" parameter to the unique name of the specific key agreement scheme instantiation (i.e., "ECDH25519" for the co-factor ECDH scheme defined in Section 4.1 and "ECDH448" for the scheme defined in Section 4.4). Note that, in this case, the "alg" name uniquely defines the curve (and, thereby, implicitly the underlying "crv" parameter).

10.3.1. Encoding of co-factor ECDH with COSE

Instantiations of co-factor ECDH used with COSE use the following encodings of inputs and outputs:

a. Curve points and private keys are encoded as specified in Section 10.1.1, where the "crv" parameter is set to the unique name of the curve used with this particular instantiation of ECDH.

When using a COSE key for this algorithm, if the "alg" field is present, it MUST be set to the (unique) name of this particular instantiation of co-factor ECDH and the "crv" parameter MUST be set to the (unique) name of the corresponding curve; if the "key_ops" field is present, it MUST include "derive shared secret" for the private key.

10.3.2. Encoding of co-factor ECDH with JOSE

Instantiations of co-factor ECDH used with JOSE use the following encodings of inputs and outputs:

a. Curve points and private keys are encoded as specified in Section 10.1.2, where the "crv" parameter is set to the unique name of the curve used with this particular instantiation of ECDH.

When using a JOSE key for this algorithm, if the "alg" field is present, it MUST be set to the (unique) name of this particular instantiation of co-factor ECDH and the "crv" parameter MUST be set to the (unique) name of the corresponding curve; if the "key_ops" field is present, it MUST include "derive shared secret" for the private key.

11. Using Wei25519 and Wei448 with PKIX and CMS

This section illustrates how to use the curves Wei25519 and Wei448 with ECDH and ECDSA with PKIX certificates (see [RFC5280] and [RFC5480]) and with CMS (see [RFC5652] and [RFC5753]).

11.1. Encoding of Short-Weierstrass Curves with PKIX

The namedCurve field in the ECParameters field in the SubjectPublicKeyInfo structure [RFC5280] indicates the elliptic curve domain parameters for a specific curve, via a unique name of the curve in question (where these are the unique object identifiers id-Wei25519 for the curve Wei25519 and id-Wei448 for the curve Wei448).

Affine and compressed curve points are encoded using the "SEC1"-representation (see Note 2 of Appendix I.8), using the tight MSB/msb-ordering conventions. This is consistent with the representation in Section 2.2 of [RFC5480], after correcting for the error in [SEC1] (for the correction, see Note in Appendix H.1).

11.2. Encoding of ECDSA Instantiations with PKIX

ECDSA25519, as defined in Section 4.3, is the instantiation of ECDSA with SHA-256 and with curve Wei25519. With [RFC5480], ECDSA can be instantiated with suitable elliptic curves and hash functions. This allows support for ECDSA25519 by instantiating ECDSA with the curve Wei25519 and the hash function SHA-256, where curve Wei25519 is identified by its object identifier id-Wei25519 (see Section 11.1), where ECDSA with SHA-256 is identified by the object identifier id-ecdsa-with-SHA256 (see [RFC5480]), and where all other aspects are specified in [RFC5480].

ECDSA448, as defined in Section 4.4, is the instantiation of ECDSA with SHAKE256 with output size d=512 bits and with curve Wei448. With [RFC5480], ECDSA can be instantiated with suitable elliptic curves and hash functions. This allows support for ECDSA448 by instantiating ECDSA with the curve Wei448 and the hash function SHAKE256 with output size of d=512 bits, where curve Wei448 is identified by its object identifier id-Wei448 (see Section 11.1), where ECDSA with SHAKE256 with output size of d=512 bits is identified by the object identifier id-ecdsa-with-shake256 (see [RFC8692]), and where all other aspects are specified in [RFC5480].

11.3. Encoding of co-factor ECDH and Other Algorithms with PKIX

With [RFC5480], the algorithm field in the SubjectPublicKeyInfo structure indicates the algorithm and the elliptic curve domain parameters for a specific curve, where that specification defines three algorithm identifiers (viz. id-ecPublicKey, id-ecDH, and id-ecMQV). Each of these algorithms can be instantiated with suitable elliptic curves, thereby allowing support for their use with the curves Wei25519 and Wei448, where these curves are identified by their unique object identifiers id-Wei25519 and id-Wei448, respectively, (see Section 11.1) and where all other aspects are specified in [RFC5480].

11.4. Encoding of Elliptic-Curve-Based Algorithms with CMS

With [RFC5753], elliptic-curve based algorithms should use one of the elliptic curve domain parameters specified in [RFC5480], where the unique name of each such curve is identified by the object identifier of this curve defined in that document. Each of these algorithms can be instantiated with suitable elliptic curves, thereby allowing support for their use with the curves Wei25519 and Wei448, where these curves are identified by their unique object identifiers id-Wei25519 and id-Wei448, respectively, (see Section 11.1) and where all other aspects are specified in [RFC5753].

12. IANA Considerations

Code points are requested for curves Wei25519 and Wei448 and their use with ECDSA and co-factor ECDH, using the representation conventions of this document.

New code points would be required in case one wishes to specify one or more other "offspring" protocols beyond those exemplified in Section 4.4. Specification hereof is, however, outside the scope of the current document.

12.1. OIDs for Use with PKIX and CMS

This section registers the following object identifiers for the curves introduced in this document:

a. id-Wei25519 OBJECT IDENTIFIER ::= TBD (Requested value: {iso(1) identified-organization(3) thawte (101) 108 });

b. id-Wei448 OBJECT IDENTIFIER ::= TBD (Requested value: {iso(1) identified-organization(3) thawte (101) 109 }).

For a description of how these are used with PKIX certificates and CMS, see Section 11.

12.2. COSE/JOSE IANA Considerations for Wei25519

12.2.1. COSE Elliptic Curves Registration

This section registers the following value in the IANA "COSE Elliptic Curves" registry [IANA.COSE.Curves].

Name: Wei25519;

Value: TBD (Requested value: -1);

Key Type: EC2 or OKP;

Description: short-Weierstrass curve Wei25519;

Change Controller: IESG;

Reference: specified in Appendix E.3 of this specification; for encodings, see Section 10.1;

Recommended: Yes.

(Note that The "kty" value for Wei25519 may be "EC2" or "OKP".)

12.2.2. COSE Algorithms Registration

This section registers the following value in the IANA "COSE Algorithms" registry [IANA.COSE.Algorithms].

Name: ECDH25519;

Value: TBD (Requested value: -24);

Description: NIST-compliant co-factor Diffie-Hellman w/ curve Wei25519 and key derivation function HKDF SHA256;

Change Controller: IESG;

Reference: specified in Section 4.1 of this specification; for encodings, see Section 10.3;

Recommended: Yes.

12.2.3. JOSE Elliptic Curves Registration

This section registers the following value in the IANA "JSON Web Key Elliptic Curve" registry [IANA.JOSE.Curves].

Curve Name: Wei25519;

Curve Description: short-Weierstrass curve Wei25519;

JOSE Implementation Requirements: Optional;

Change Controller: IESG;

Reference: specified in Appendix E.3 of this specification; for encodings, see Section 10.1.

(Note that The "kty" value for Wei25519 may be "EC" or "OKP".)

12.2.4. JOSE Algorithms Registration (1/2)

This section registers the following value in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms].

Algorithm Name: ECDSA25519;

Algorithm Description: ECDSA using SHA-256 and curve Wei25519;

Algorithm Usage Locations: alg;

JOSE Implementation Requirements: Optional;

Change Controller: IESG;

Reference: specified in Section 4.3 of this specification; for encodings, see Section 10.2;

Algorithm Analysis Document(s): Section 4.3 of this specification.

12.2.5. JOSE Algorithms Registration (2/2)

This section registers the following value in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms].

Algorithm Name: ECDH25519;

Algorithm Description: NIST-compliant co-factor Diffie-Hellman w/ curve Wei25519 and key derivation function HKDF SHA256;

Algorithm Usage Locations: alg;

JOSE Implementation Requirements: Optional;

Change Controller: IESG;

Reference: specified in Section 4.1 of this specification; for encodings, see Section 10.3;

Algorithm Analysis Document(s): Section 4.1 of this specification.

12.3. COSE/JOSE IANA Considerations for Wei448

12.3.1. COSE Elliptic Curves Registration

This section registers the following value in the IANA "COSE Elliptic Curves" registry [IANA.COSE.Curves].

Name: Wei448;

Value: TBD (Requested value: -2);

Key Type: EC2 or OKP;

Description: short-Weierstrass curve Wei448;

Change Controller: IESG;

Reference: specified in Appendix M.3 of this specification; for encodings, see Section 10.1;

Recommended: Yes.

(Note that The "kty" value for Wei448 may be "EC2" or "OKP".)

12.3.2. COSE Algorithms Registration (1/2)

This section registers the following value in the IANA "COSE Algorithms" registry [IANA.COSE.Algorithms].

Name: ECDSA with SHAKE256;

Value: TBD (Requested value: -48);

Description: ECDSA with SHAKE256;

Change Controller: IESG;

Reference: specified in Section 4.4 of this specification; for encodings, see Section 10.2;

Recommended: Yes.

12.3.3. COSE Algorithms Registration (2/2)

This section registers the following value in the IANA "COSE Algorithms" registry [IANA.COSE.Algorithms].

Name: ECDH448;

Value: TBD (Requested value: -49);

Description: NIST-compliant co-factor Diffie-Hellman w/ curve Wei448 and key derivation function HKDF SHA512;

Change Controller: IESG;

Reference: specified in Section 4.4 of this specification; for encodings, see Section 10.1; for key derivation, see Section 11.1 of [RFC8152];

Recommended: Yes.

12.3.4. JOSE Elliptic Curves Registration

This section registers the following value in the IANA "JSON Web Key Elliptic Curve" registry [IANA.JOSE.Curves].

Curve Name: Wei448;

Curve Description: short-Weierstrass curve Wei448;

JOSE Implementation Requirements: Optional;

Change Controller: IESG;

Reference: specified in Appendix M.3 of this specification; for encodings, see Section 10.1.

(Note that The "kty" value for Wei448 may be "EC" or "OKP".)

12.3.5. JOSE Algorithms Registration (1/2)

This section registers the following value in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms].

Algorithm Name: ECDSA448;

Algorithm Description: ECDSA using SHAKE256 and curve Wei448;

Algorithm Usage Locations: alg;

JOSE Implementation Requirements: Optional;

Change Controller: IESG;

Reference: specified in Section 4.4 of this specification; for encodings, see Section 10.2;

Algorithm Analysis Document(s): Section 4.4 of this specification.

12.3.6. JOSE Algorithms Registration (2/2)

This section registers the following value in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms].

Algorithm Name: ECDH448;

Algorithm Description: NIST-compliant co-factor Diffie-Hellman w/ curve Wei448;

Algorithm Usage Locations: alg;

JOSE Implementation Requirements: Optional;

Change Controller: IESG;

Reference: specified in Section 4.4 of this specification; for encodings, see Section 10.3;

Algorithm Analysis Document(s): Section 4.4 of this specification.

13. Acknowledgements

Thanks to Nikolas Rosener for discussions surrounding implementation details of the techniques described in this document and to Phillip Hallam-Baker for triggering inclusion of verbiage on the use of Montgomery ladders with recovery of the y-coordinate. Thanks to Stanislav Smyshlyaev, Vasily Nikolaev, and Benjamin Smith for their careful reviews.

14. References

14.1. Normative References

[ANSI-X9.62]
ANSI X9.62-2005, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", American National Standard for Financial Services, Accredited Standards Committee X9, Inc, Anapolis, MD, 2005.

[FIPS-180-4]
FIPS 180-4, "Secure Hash Standard (SHS), Federal Information Processing Standards Publication 180-4", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, August 2015.

[FIPS-186-4]
FIPS 186-4, "Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-4", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, July 2013.

[FIPS-202]
FIPS 202, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, Federal Information Processing Standards Publication 202", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, August 2015.

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[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/info/rfc5639>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, DOI 10.17487/RFC5753, January 2010, <https://www.rfc-editor.org/info/rfc5753>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

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

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

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

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/info/rfc8037>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

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

[RFC8692] Kampanakis, P. and Q. Dang, "Internet X.509 Public Key Infrastructure: Additional Algorithm Identifiers for RSASSA-PSS and ECDSA Using SHAKEs", RFC 8692, DOI 10.17487/RFC8692, December 2019, <https://www.rfc-editor.org/info/rfc8692>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[SEC1] SEC1, "SEC 1: Elliptic Curve Cryptography, Version 2.0", Standards for Efficient Cryptography, , June 2009.

[SEC2] SEC2, "SEC 2: Elliptic Curve Cryptography, Version 2.0", Standards for Efficient Cryptography, , January 2010.

[SP-800-56a]
NIST SP 800-56a, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Log Cryptography, Revision 3", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, April 2018.

[SP-800-56c]
NIST SP 800-56c, "Recommendation for Key-Derivation Methods in Key-Establishment Schemes, Revision 1", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, April 2018.

14.2. Informative References

[comm-FIPS-186-5]
FIPS 186-5, "Public Comments Received on Draft FIPS Pub 186-5", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, April 6, 2020.

[draft-FIPS-186-5]
FIPS 186-5, "Digital Signature Standard (DSS) (Draft)", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, October 31, 2019.

[draft-NIST-800-186]
NIST SP 800-186, "Recommendations for Discrete Logarithm-Based Cryptography, Elliptic Curve Domain Parameters (Draft)", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, October 31, 2019.

[ECC] I.F. Blake, G. Seroussi, N.P. Smart, "Elliptic Curves in Cryptography", Cambridge University Press, Lecture Notes Series 265, July 1999.

[ECC-Isogeny]
E. Brier, M. Joye, "Fast Point Multiplication on Elliptic Curves through Isogenies", AAECC, Lecture Notes in Computer Science, Vol. 2643, New York: Springer-Verlag, 2003.

[FIPS-140-2]
FIPS 140-2, "Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, August 28, 2020.

[GECC] D. Hankerson, A.J. Menezes, S.A. Vanstone, "Guide to Elliptic Curve Cryptography", New York: Springer-Verlag, 2004.

[Handbook] A.J. Menezes, P. van Oorschot, S.A. Vanstone,, "Handbook of Applied Cryptography", Boca Raton: CRC Press, 1995.

[IANA.COSE.Algorithms] IANA, "COSE Algorithms", IANA, https://www.iana.org/assignments/cose/cose.xhtml#algorithms.

[IANA.COSE.Curves] IANA, "COSE Elliptic Curves", IANA, https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves.

[IANA.JOSE.Algorithms] IANA, "JSON Web Signature and Encryption Algorithms", IANA, https://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms.

[IANA.JOSE.Curves] IANA, "JSON Web Key Elliptic Curve", IANA, https://www.iana.org/assignments/jose/jose.xhtml#web-key-elliptic-curve.

[Mont-Ladder] P.L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization", Mathematics of Computation, Vol. 48, 1987.

[RFC8928] Thubert, P., Ed., Sarikaya, B., Sethi, M., and R. Struik, "Address-Protected Neighbor Discovery for Low-Power and Lossy Networks", RFC 8928, DOI 10.17487/RFC8928, November 2020, <https://www.rfc-editor.org/info/rfc8928>.

[Rosener] N. Rosener, "Evaluating the Performance of Transformations Between Curve Representations in Elliptic Curve Cryptography for Constrained Device Security", M.Sc. Universitat Bremen, August 2018.

[SWUmap] E. Brier, J-S. Coron, Th. Icart, D. Madore, H. Randriam, M. Tibouchi, "Efficient Indifferentiable Hashing into Ordinary Elliptic Curves", CRYPTO 2010, Lecture Notes in Computer Science, Vol. 6223, New York: Springer-Verlag, 2010.

[tEd] D.J. Bernstein, P. Birkner, M. Joye, T. Lange, C. Peters, "Twisted Edwards Curves", Africacrypt 2008, Lecture Notes in Computer Science, Vol. 5023, New York: Springer-Verlag, 2008.

[tEd-Formulas] H. Hisil, K.K.H. Wong, G. Carter, E. Dawson, "Twisted Edwards Curves Revisited", ASIACRYPT 2008, Lecture Notes in Computer Science, Vol. 5350, New York: Springer-Verlag, 2008.

[Tibouchi] M. Tibouchi, "Elligator Squared -- Uniform Points on Elliptic Curves of Prime Order as Uniform Random Strings", Financial Cryptography 2014, Lecture Notes in Computer Science, Vol. 8437, New York: Springer-Verlag, 2014.

[Tibouchi-cleancut] T. Kim, M. Tibouchi, "Improved Elliptic Curve Hashing and Point Representation", DCC 2017, Des. Codes Cryptogr., Vol. 82, pp. 161-177, New York: Springer-Verlag, 2017.

[Wei-Ladder] T. Izu, Ts. Takagi,, "A Fast Parallel Elliptic Curve Multiplication Resistant Against Side Channel Attacks", Centre for Applied Cryptographic Research, Corr 2002-03, 2002.

Appendix A. Some (Non-Binary) Elliptic Curves

This section defines the three different curve models we consider, viz. short-Weierstrass curves, Montgomery curves, and twisted Edwards curves. For nomenclature, see Appendix B.

A.1. Curves in Short-Weierstrass Form

Let GF(q) denote the finite field with q elements, where q is an odd prime power and where q is not divisible by three. Let W_{a,b} be the Weierstrass curve with defining equation Y^2 = X^3 + a*X + b, where a and b are elements of GF(q) and where 4*a^3 + 27*b^2 is nonzero. The points of W_{a,b} are the ordered pairs (X, Y) whose coordinates are elements of GF(q) and that satisfy the defining

equation (the so-called affine points), together with the special point O (the so-called "point at infinity"). This set forms a group under addition, via the so-called "chord-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix C.1 for details of the group operation.

A quadratic twist of W_{a,b} is a curve W_{a',b'} defined over the same field for which a':= a*gamma^2 and b':=b*gamma^3, where gamma is an element of GF(q) that is not a square in GF(q).

A.2. Montgomery Curves

Let GF(q) denote the finite field with q elements, where q is an odd prime power. Let M_{A,B} be the Montgomery curve with defining equation B*v^2 = u^3 + A*u^2 + u, where A and B are elements of GF(q) and where A is unequal to (+/-)2 and where B is nonzero. The points of M_{A,B} are the ordered pairs (u, v) whose coordinates are elements of GF(q) and that satisfy the defining equation (the so-called affine points), together with the special point O (the so-called "point at infinity"). This set forms a group under addition, via the so-called "chord-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix C.2 for details of the group operation.

A quadratic twist of M_{A,B} is a curve M_{A',B'} defined over the same field for which A':= A and B':=B*gamma, where gamma is an element of GF(q) that is not a square in GF(q).

A.3. Twisted Edwards Curves

Let GF(q) denote the finite field with q elements, where q is an odd prime power. Let E_{a,d} be the twisted Edwards curve with defining equation a*x^2 + y^2 = 1+ d*x^2*y^2, where a and d are distinct nonzero elements of GF(q). The points of E_{a,d} are the ordered pairs (x, y) whose coordinates are elements of GF(q) and that satisfy the defining equation (the so-called affine points). It can be shown that this set forms a group under addition if a is a square in GF(q), whereas d is not, where the point O:=(0, 1) serves as the identity element. (Note that the identity element satisfies the defining equation.) See Appendix C.3 for details of the group operation. (All curves E_{a,d} in this document are assumed to satisfy the condition on domain parameters a and d above and, thereby, satisfy the Note in that appendix.)

An Edwards curve is a twisted Edwards curve with a=1.

A quadratic twist of E_{a,d} is a curve E_{a',d'} defined over the same field for which a':= a*gamma and d':=d*gamma, where gamma is an element of GF(q) that is not a square in GF(q).

Appendix B. Elliptic Curve Nomenclature and Finite Fields

This section provides brief background information on elliptic curves and finite fields that should be sufficient to understand constructions and examples in this document.

B.1. Elliptic Curve Nomenclature

The set of points of each curve defined in Appendix A forms a commutative group under addition (denoted by '+'). In Appendix C we specify the group laws, which depend on the curve model in question. For completeness, we here include some common elliptic curve nomenclature and basic properties (primarily so as to keep this document self-contained). These notions are mainly used in Appendix E and Appendix G and not essential for our exposition. This section can be skipped at first reading.

Any point P of a curve E is a generator of the cyclic subgroup <P>:={k*P | k = 0, 1, 2,...} of the curve. (Here, k*P denotes the sum of k copies of P, where 0*P is the identity element O of the curve; k*P is commonly referred to as scalar multiplication of P by k.) If <P> has cardinality l, then l is called the order of P and l is the smallest positive integer so that l*P=O. The order of curve E is the cardinality of the set of its points, commonly denoted by |E|. A curve is cyclic if it is generated by some point of this curve. All curves of prime order are cyclic, while all curves of order h*n, where n is a large prime number and where h is a small number (the so-called co-factor), have a large cyclic subgroup of prime order n. In this case, a generator of order n is called a base point, commonly denoted by G, while a point of order dividing h is said to be in the small subgroup (or said to be a low-order point). For curves of prime order, this small subgroup is the singleton set, consisting of only the identity element O. A point that is not in the small subgroup is said to be a high-order point (since it has order at least n). A point P of the curve is in the small subgroup if h*P=O (and is a high-order point otherwise); this point P has order n if n*P=O and if it is not the identity element O. (The latter order check is commonly called full public key validation.) The above definitions extend to curves with a relatively large co-factor, by defining n to be the size of its largest prime-order subgroup.

If R is a point of the curve that is also contained in <P>, there is a unique integer k in the interval [0, l-1] so that R=k*P, where l is the order of P. This number is called the discrete logarithm of R to

the base P. The discrete logarithm problem is the problem of finding the discrete logarithm of R to the base P for any two points P and R of the curve, if such a number exists.

Random points R of <P>, where P has order l, can be computed by generating a random integer k in the interval [0, l-1] and by subsequently computing R:=k*P, where R then has order l/gcd(k,l). In particular, if P is a high-order point (of curve E of order h*n), then so is R, unless k is a multiple of n (in which case R is a low-order point). For methods for generating k, see Appendix P.

If P is a fixed base point G of the curve, the pair (k, R:=k*G) is commonly called a public-private key pair, the integer k the private key, and the point R the corresponding public key. The private key k can be represented as an integer in the interval [0,n-1], where G has order n. If this representation is nonzero, R has order n; otherwise, it has order one and is the identity element O of the curve.

A curve E defined over the field GF(q) has order |E| relatively close to q. More precisely, |E|=q+1-t for some integer t (the so-called trace) with absolute value at most 2*|sqrt(q)|. This is commonly referred to as the Hasse bound.

In this document, a quadratic twist of a curve E defined over a field GF(q) is a specific curve E' related to E defined over the same field, with cardinality |E'|, where |E|+|E'|=2*(q+1). If E is a curve in one of the curve models specified in this document, a quadratic twist E' of this curve can be expressed using the same curve model, although (naturally) with its own curve parameters (see Appendix A). Points that are points of both E and E' have order one or two. Two curves E1 and E2 defined over the field GF(q) are said to be isogenous if these have the same order and are said to be isomorphic if the defining equation of E1 can be transformed into the defining equation of E2 via a so-called admissible change of variables. Note that isomorphic curves have necessarily the same order and are, thus, a special case of isogenous curves. Isomorphic curves have the same group structure, whereas this is not necessarily the case for isogenous curves. Further details are out of scope.

Curves in short-Weierstrass form can have prime order, whereas Montgomery curves and twisted Edwards curves always have an order that is a multiple of four (and, thereby, a small subgroup of cardinality four).

An ordered pair (x, y) whose coordinates are elements of GF(q) can be associated with any ordered triple of the form [x*z: y*z: z], where z is a nonzero element of GF(q), and can be uniquely recovered from

such a representation. The latter representation is commonly called a representation in projective coordinates. Sometimes, yet other representations are useful (e.g., representation in Jacobian coordinates). Further details are out of scope.

The group laws in Appendix C are mostly expressed in terms of affine points, but can also be expressed in terms of the representation of these points in projective coordinates, thereby allowing clearing of denominators. The group laws may also involve non-affine points (such as the point at infinity O of a Weierstrass curve or of a Montgomery curve). Those can also be represented in projective coordinates. Further details are out of scope.

B.2. Finite Fields

The field GF(q), where q is a prime power, is defined as follows.

If q:=p is a prime number, the field GF(p) consists of the integers in the interval [0,p-1] and two binary operations on this set: addition and multiplication modulo p. This field is commonly called a prime field. The additive and multiplicative identity elements are 0 and 1, respectively.

If q:=p^m, where p is a prime number and where m>0, the field GF(q) is defined in terms of an irreducible polynomial f(z) in z of degree m with coefficients in GF(p) (i.e., f(z) cannot be written as the product of two polynomials in z of lower degree with coefficients in GF(p)): in this case, GF(q) consists of the polynomials in z of degree smaller than m with coefficients in GF(p) and two binary operations on this set: polynomial addition and polynomial multiplication modulo the irreducible polynomial f(z). By definition, each element x of GF(q) is a polynomial in z of degree smaller than m and can, therefore, be uniquely represented as a vector (x_{m-1}, x_{m-2}, ..., x_1, x_0) of length m with coefficients in GF(p), where x_i is the coefficient of z^i of polynomial x. Note that this representation depends on the irreducible polynomial f(z) of the field GF(p^m) in question (which is often fixed in practice). Note that GF(q) contains the prime field GF(p) as a subset. If m=1, the definitions of GF(p) and GF(p^1) above coincide, since each nonzero element of GF(p) can be viewed as a polynomial in z of degree zero. If m>1 (i.e., if q is a strict prime power), then GF(q) is called a (nontrivial) extension field of GF(p). The number p is called the characteristic of GF(q).

Any nonzero element g of GF(q) is a generator of the cyclic multiplicative subgroup <g>:={g^k | k = 0, 1, 2,...} of GF(q)\{0}. (Here, g^k denotes the product of k copies of g, where g^0 is the multiplicative identity element 1 of GF(q)\{0}.) If <g> has

cardinality l, then l is called the order of g and l is the smallest positive integer so that g^l=1. For each finite field GF(q), the set GF(q)\{0} forms a cyclic group, i.e., it is generated by some nonzero element hereof. Each such generator is called a primitive element of GF(q) and has order q-1. Each nonzero element of GF(q) has order dividing q-1 (a property commonly referred to as Fermat's Little Theorem).

A field element y is called a square in GF(q) if it can be expressed as y:=x^2 for some x in GF(q); it is called a non-square in GF(q) otherwise. If y is a square in GF(q), we denote by sqrt(y) one of its square roots (the other one being -sqrt(y)). For methods for computing square roots in GF(q) - if these exist - and for computing inverses in GF(q)\{0}, see Appendix K.1 and Appendix K.2, respectively. For methods for mapping a nonzero field element that is not a square in GF(q) to a point of a curve, see Appendix K.3 (or see Appendix K.4, if one wishes to always obtain a high-order point of the curve in question).

NOTE: The curves in Appendix E and Appendix G are all defined over a prime field GF(p), thereby reducing all operations to simple modular integer arithmetic. Strictly speaking we could, therefore, have refrained from introducing extension fields. Nevertheless, we included the more general exposition, so as to accommodate potential introduction of new curves that are defined over a (nontrivial) extension field at some point in the future. This includes curves proposed for post-quantum isogeny-based schemes, which are defined over a quadratic extension field (i.e., where q:=p^2), and elliptic curves used with pairing-based cryptography. The exposition in either case is almost the same and now automatically yields, e.g., data conversion routines for any finite field object (see Appendix I). Readers not interested in this could simply view all fields as prime fields.

Appendix C. Elliptic Curve Group Operations

This section specifies group operations for elliptic curves in short-Weierstrass form, for Montgomery curves, and for twisted Edwards curves.

C.1. Group Laws for Weierstrass Curves

For each point P of the Weierstrass curve W_{a,b}, the point at infinity O serves as identity element, i.e., P + O = O + P = P.

For each affine point P:=(X, Y) of the Weierstrass curve W_{a,b}, the point -P is the point (X, -Y) and one has P + (-P) = O (i.e., -P is the inverse of P). For the point at infinity O, one has -O:=O.

Let P1:=(X1, Y1) and P2:=(X2, Y2) be distinct affine points of the Weierstrass curve W_{a,b} and let Q:=P1 + P2, where Q is not the identity element. Then Q=(X, Y), where

X + X1 + X2 = lambda^2 and Y + Y1 = lambda*(X1 - X), where

lambda:= (Y2 - Y1)/(X2 - X1).

Let P:=(X1, Y1) be an affine point of the Weierstrass curve W_{a,b} and let Q:=2*P, where Q is not the identity element. Then Q=(X, Y), where

X + 2*X1 = lambda^2 and Y + Y1 = lambda*(X1 - X), where

lambda:=(3*X1^2 + a)/(2*Y1).

From the group laws above it follows that if P=(X, Y), P1=(X1, Y1), and P2=(X2, Y2) are distinct affine points of the Weierstrass curve W_{a,b} with P2:=P+P1 and if Y is nonzero, then the Y-coordinate of P1 can be expressed in terms of the X-coordinates of P, P1, and P2, and the Y-coordinate of P, since

2*Y*Y1=(X*X1+a)*(X+X1)+2*b-X2*(X-X1)^2.

This property allows recovery of the Y-coordinate of a point P1=k*P that is computed via the so-called Montgomery ladder, where P is an affine point with nonzero Y-coordinate (i.e., it does not have order two). For future reference, note that the expression above uniquely determines the X-coordinate of P2 in terms of the X-coordinates of P and P1 and the product of their Y-coordinates. Further details are out of scope.

C.2. Group Laws for Montgomery Curves

For each point P of the Montgomery curve M_{A,B}, the point at infinity O serves as identity element, i.e., P + O = O + P = P.

For each affine point P:=(u, v) of the Montgomery curve M_{A,B}, the point -P is the point (u, -v) and one has P + (-P) = O (i.e., -P is the inverse of P). For the point at infinity O, one has -O:=O.

Let P1:=(u1, v1) and P2:=(u2, v2) be distinct affine points of the Montgomery curve M_{A,B} and let Q:=P1 + P2, where Q is not the identity element. Then Q=(u, v), where

u + u1 + u2 = B*lambda^2 - A and v + v1 = lambda*(u1 - u), where

lambda:=(v2 - v1)/(u2 - u1).

Let P:=(u1, v1) be an affine point of the Montgomery curve M_{A,B} and let Q:=2*P, where Q is not the identity element. Then Q=(u, v), where

u + 2*u1 = B*lambda^2 - A and v + v1 = lambda*(u1 - u), where

lambda:=(3*u1^2 + 2*A*u1+1)/(2*B*v1).

From the group laws above it follows that if P=(u, v), P1=(u1, v1), and P2=(u2, v2) are distinct affine points of the Montgomery curve M_{A,B} with P2:=P+P1 and if v is nonzero, then the v-coordinate of P1 can be expressed in terms of the u-coordinates of P, P1, and P2, and the v-coordinate of P, since

2*B*v*v1=(u*u1+1)*(u+u1+2*A)-2*A-u2*(u-u1)^2.

This property allows recovery of the v-coordinate of a point P1=k*P that is computed via the so-called Montgomery ladder, where P is an affine point with nonzero v-coordinate (i.e., it does not have order two). For future reference, note that the expression above uniquely determines the u-coordinate of P2 in terms of the u-coordinates of P and P1 and the product of their v-coordinates. Further details are out of scope.

C.3. Group Laws for Twisted Edwards Curves

Note: The group laws below hold for twisted Edwards curves E_{a,d} where a is a square in GF(q), whereas d is not. In this case, the addition formulae below are defined for each pair of points, without exceptions. Generalizations of this group law to other twisted Edwards curves are out of scope.

For each point P of the twisted Edwards curve E_{a,d}, the point O:=(0,1) serves as identity element, i.e., P + O = O + P = P.

For each point P:=(x, y) of the twisted Edwards curve E_{a,d}, the point -P is the point (-x, y) and one has P + (-P) = O (i.e., -P is the inverse of P).

Let P1:=(x1, y1) and P2:=(x2, y2) be points of the twisted Edwards curve E_{a,d} and let Q:=P1 + P2. Then Q=(x, y), where

x = (x1*y2 + x2*y1)/(1 + d*x1*x2*y1*y2) and

y = (y1*y2 - a*x1*x2)/(1 - d*x1*x2*y1*y2).

Let P:=(x1, y1) be a point of the twisted Edwards curve E_{a,d} and let Q:=2*P. Then Q=(x, y), where

x = (2*x1*y1)/(1 + d*x1^2*y1^2) and

y = (y1^2 - a*x1^2)/(1 - d*x1^2*y1^2).

Note that one can use the formulae for point addition for point doubling, taking inverses, and adding the identity element as well (i.e., the point addition formulae are uniform and complete (subject to our Note above)).

From the group laws above (subject to our Note above) it follows that if P=(x, y), P1=(x1, y1), and P2=P=(x2, y2) are points of the twisted Edwards curve E_{a,d} with P2:=P+P1 and if x is nonzero, then the x-coordinate of P1 can be expressed in terms of the y-coordinates of P, P1, and P2, and the x-coordinate of P, since

x*x1*(a-d*y*y1*y2)=y*y1-y2.

(Here, observe that a-d*y*y1*y2 is nonzero per our Note above.) This property allows recovery of the x-coordinate of a point P1=k*P that is computed via the so-called Montgomery ladder, where P is an affine point with nonzero x-coordinate (i.e., it does not have order one or two). For future reference, note that the group law (subject to our Note above) uniquely determines the y-coordinate of P2 in terms of the y-coordinates of P and P1 and the product of their x-coordinates. Further details are out of scope.

Appendix D. Relationships Between Curve Models

The non-binary curves specified in Appendix A are expressed in different curve models, viz. as curves in short-Weierstrass form, as Montgomery curves, or as twisted Edwards curves. These curve models are related, as follows.

D.1. Mapping between Twisted Edwards Curves and Montgomery Curves

One can map points of the Montgomery curve M_{A,B} to points of the twisted Edwards curve E_{a,d}, where a:=(A+2)/B and d:=(A-2)/B and, conversely, map points of the twisted Edwards curve E_{a,d} to points of the Montgomery curve M_{A,B}, where A:=2*(a+d)/(a-d) and where B:=4/(a-d). For twisted Edwards curves we consider (i.e., those where a is a square in GF(q), whereas d is not), this defines a one-to-one correspondence, which - in fact - is an isomorphism between M_{A,B} and E_{a,d}, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

For the Montgomery curves and twisted Edwards curves we consider, the mapping from M_{A,B} to E_{a,d} is defined by mapping the point at infinity O and the point (0, 0) of order two of M_{A,B} to,

respectively, the point (0, 1) and the point (0, -1) of order two of E_{a,d}, while mapping each other point (u, v) of M_{A,B} to the point (x,y):=(u/v,(u-1)/(u+1)) of E_{a,d}. (Note that this is well-defined, since neither (A-2)/B nor A^2-4 are squares in GF(q), so M_{A,B} has a single point of order two and no affine points (u,v) with u=-1.) The inverse mapping from E_{a,d} to M_{A,B} is defined by mapping the point (0, 1) and the point (0, -1) of order two of E_{a,d} to, respectively, the point at infinity O and the point (0, 0) of order two of M_{A,B}, while each other point (x, y) of E_{a,d} is mapped to the point (u,v):=((1+y)/(1-y),(1+y)/((1-y)*x)) of M_{A,B}. (Note that this is well-defined, since for points (x,y) of E_{a,d}, x=0 only if y=(+/-)1.)

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a twisted Edwards curve on the corresponding Montgomery curve, or vice-versa, and translating the result back to the original curve, thereby potentially allowing code reuse.

D.2. Mapping between Montgomery Curves and Weierstrass Curves

One can map points of the Montgomery curve M_{A,B} to points of the Weierstrass curve W_{a,b}, where a:=(3-A^2)/(3*B^2) and b:=(2*A^3-9*A)/(27*B^3). This defines a one-to-one correspondence, which - in fact - is an isomorphism between M_{A,B} and W_{a,b}, thereby showing that, e.g., the discrete logarithm problem in either curve model is equally hard.

The mapping from M_{A,B} to W_{a,b} is defined by mapping the point at infinity O of M_{A,B} to the point at infinity O of W_{a,b}, while mapping each other point (u,v) of M_{A,B} to the point (X,Y):=((u+A/3)/B,v/B) of W_{a,b}.

Note that not all Weierstrass curves can be mapped to Montgomery curves, since the latter have a point of order two and the former may not. In particular, if a Weierstrass curve has prime order, such as is the case with the so-called NIST prime curves, this inverse mapping is not defined.

If the Weierstrass curve W_{a,b} has a point (alpha,0) of order two and c:=a+3*(alpha)^2 is a square in GF(q), one can map points of this curve to points of the Montgomery curve M_{A,B}, where A:=3*alpha/gamma and B:=1/gamma and where gamma is any square root of c. In this case, the mapping from W_{a,b} to M_{A,B} is defined by mapping the point at infinity O of W_{a,b} to the point at infinity O of M_{A,B}, while mapping each other point (X,Y) of W_{a,b} to the point (u,v):=((X-alpha)/gamma,Y/gamma) of M_{A,B}. As before, this defines a one-to-one correspondence, which - in fact - is an isomorphism

between W_{a,b} and M_{A,B}. It is easy to see that the mapping from W_{a,b} to M_{A,B} and that from M_{A,B} to W_{a,b} (if defined) are each other's inverse.

This mapping can be used to implement elliptic curve group operations originally defined for a twisted Edwards curve or for a Montgomery curve using group operations for the corresponding elliptic curve in short-Weierstrass form and translating the result back to the original curve, thereby potentially allowing code reuse.

Note that implementations for elliptic curves with short-Weierstrass form that hard-code the domain parameter a to a= -3 (which value is known to allow more efficient implementations) cannot always be used this way, since the curve W_{a,b} resulting from an isomorphic mapping cannot always be expressed as a Weierstrass curve with a=-3 via a coordinate transformation. For more details, see Appendix F.

D.3. Mapping between Twisted Edwards Curves and Weierstrass Curves

One can map points of the twisted Edwards curve E_{a,d} to points of the Weierstrass curve W_{a,b}, via function composition, where one uses the isomorphic mapping between twisted Edwards curves and Montgomery curves of Appendix D.1 and the one between Montgomery and Weierstrass curves of Appendix D.2. Obviously, one can use function composition (now using the respective inverses - if these exist) to realize the inverse of this mapping.

Appendix E. Curve25519 and Cousins

This section introduces curves related to Curve25519 and explains their relationships.

E.1. Curve Definition and Alternative Representations

The elliptic curve Curve25519 is the Montgomery curve M_{A,B} defined over the prime field GF(p), with p:=2^{255}-19, where A:=486662 and B:=1. This curve has order h*n, where h=8 and where n is a prime number. For this curve, A^2-4 is not a square in GF(p), whereas A+2 is. The quadratic twist of this curve has order h1*n1, where h1=4 and where n1 is a prime number. For this curve, the base point is the point (Gu, Gv), where Gu=9 and where Gv is an odd integer in the interval [0, p-1].

This curve has the same group structure as (is "isomorphic" to) the twisted Edwards curve E_{a,d} defined over GF(p), with as base point the point (Gx, Gy), where parameters are as specified in Appendix E.3. This curve is denoted as Edwards25519. For this

curve, the parameter a is a square in GF(p), whereas d is not, so the group laws of Appendix C.3 apply.

The curve is also isomorphic to the elliptic curve W_{a,b} in short-Weierstrass form defined over GF(p), with as base point the point (GX, GY), where parameters are as specified in Appendix E.3. This curve is denoted as Wei25519. For this curve, the parameter b is a square in GF(p). (For future reference, we note that this curve has no affine points with x-coordinate -1.)

E.2. Switching between Alternative Representations

Each affine point (u, v) of Curve25519 corresponds to the point (X, Y):=(u + A/3, v) of Wei25519, while the point at infinity of Curve25519 corresponds to the point at infinity of Wei25519. (Here, we used the mappings of Appendix D.2 and that B=1.) Under this mapping, the base point (Gu, Gv) of Curve25519 corresponds to the base point (GX, GY) of Wei25519. The inverse mapping maps the affine point (X, Y) of Wei25519 to (u, v):=(X - A/3, Y) of Curve25519, while mapping the point at infinity of Wei25519 to the point at infinity of Curve25519. Note that this mapping involves a simple shift of the first coordinate and can be implemented via integer-only arithmetic as a shift of delta for the isomorphic mapping and a shift of -delta for its inverse, where delta:=(p+A)/3 is the integer defined by

delta 19298681539552699237261830834781317975544997444273427339909597334652188435537

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaad2451).

(Note that, depending on the implementation details of the field arithmetic, one may have to shift the result by +p or -p if this integer is not in the interval [0,p-1].)

The curve Edwards25519 is isomorphic to the curve Curve25519, where the base point (Gu, Gv) of Curve25519 corresponds to the base point (Gx,Gy) of Edwards25519 and where the point at infinity and the point (0,0) of order two of Curve25519 correspond to, respectively, the point (0, 1) and the point (0, -1) of order two of Edwards25519 and where each other point (u, v) of Curve25519 corresponds to the point (c*u/v, (u-1)/(u+1)) of Edwards25519, where c is the element of GF(p) defined by

c sqrt(-(A+2)/B)

510425693991605361302061352331463292841522022530346318226818337886668772152 07

(=0x70d9120b 9f5ff944 2d84f723 fc03b081 3a5e2c2e b482e57d
3391fb55 00ba81e7).

(Here, we used the mapping of Appendix D.1 and normalized this using the mapping of Appendix F.1 (where the element s of that appendix is set to c above).) The inverse mapping from Edwards25519 to Curve25519 is defined by mapping the point (0, 1) and the point (0, -1) of order two of Edwards25519 to, respectively, the point at infinity and the point (0,0) of order two of Curve25519 and having each other point (x, y) of Edwards25519 correspond to the point ((1 + y)/(1 - y), c*(1 + y)/((1-y)*x)) of Curve25519.

The curve Edwards25519 is isomorphic to the Weierstrass curve Wei25519, where the base point (Gx, Gy) of Edwards25519 corresponds to the base point (GX,GY) of Wei25519 and where the identity element (0,1) and the point (0,-1) of order two of Edwards25519 correspond to, respectively, the point at infinity O and the point (A/3, 0) of order two of Wei25519 and where each other point (x, y) of Edwards25519 corresponds to the point (X, Y):=((1+y)/(1-y)+A/3, c*(1+y)/((1-y)*x)) of Wei25519, where c was defined before. (Here, we used the mapping of Appendix D.3.) The inverse mapping from Wei25519 to Edwards25519 is defined by mapping the point at infinity O and the point (A/3, 0) of order two of Wei25519 to, respectively, the identity element (0,1) and the point (0,-1) of order two of Edwards25519 and having each other point (X, Y) of Wei25519 correspond to the point (c*(X-A/3)/Y, (X-A/3-1)/(X-A/3+1)) of Edwards25519.

Note that these mappings can be easily realized if points are represented in projective coordinates, using a few field multiplications only, thus allowing switching between alternative curve representations with negligible relative incremental cost.

E.3. Domain Parameters

The parameters of the Montgomery curve and the corresponding isomorphic curves in twisted Edwards curve and short-Weierstrass form are as indicated below. Here, the domain parameters of the Montgomery curve Curve25519 and of the twisted Edwards curve Edwards25519 are as specified in [RFC7748]; the domain parameters of Wei25519 are "new".

General parameters (for all curve models):

p 2^{255}-19

(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffed)

h 8

n 7237005577332262213973186563042994240857116359379907606001950938285454250989

(=2^{252} + 0x14def9de a2f79cd6 5812631a 5cf5d3ed)

h1 4

n1 1447401115466452442794637312608598848160326344765032579786049412540737390799 7

(=2^{253} - 0x29bdf3bd 45ef39ac b024c634 b9eba7e3)

Montgomery curve-specific parameters (for Curve25519):

A 486662 (=0x076d06)

B 1 (=0x01)

Gu 9 (=0x09)

Gv 1478161944758954479102059356840998688726460613461647528896488183775558623740 1

(=0x20ae19a1 b8a086b4 e01edd2c 7748d14c 923d4d7e 6d7c61b2 29e9c5a2 7eced3d9)

Twisted Edwards curve-specific parameters (for Edwards25519):

a -1 (-0x01)

d -121665/121666 = - (A-2)/(A+2)

(=3709570593466943934313808350875456518954211387984321901638878553308594028355 5)

(=0x52036cee 2b6ffe73 8cc74079 7779e898 00700a4d 4141d8ab 75eb4dca 135978a3)

Gx 1511222134953540077250115140958853151145401269304185720604611328394984776220 2

(=0x216936d3 cd6e53fe c0a4e231 fdd6dc5c 692cc760 9525a7b2 c9562d60 8f25d51a)

Gy 4/5

(=463168356949264781694283940034751631413079938662562256157830336
03165251855960)

(=0x66666666 66666666 66666666 66666666 66666666 66666666
66666666 66666658)

Weierstrass curve-specific parameters (for Wei25519):

a 192986815395526992372618308347813179755449974442734273399095973345
73241639236

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaa98 4914a144)

b 557517466698189089076452890782571408182411037279010123152944008375
56729358436

(=0x7b425ed0 97b425ed 097b425e d097b425 ed097b42 5ed097b4
260b5e9c 7710c864)

GX 192986815395526992372618308347813179755449974442734273399095973345
52188435546

(=0x2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaaaa aaad245a)

GY 147816194475895447910205935684099868872646061346164752889648818375
55586237401

(=0x20ae19a1 b8a086b4 e01edd2c 7748d14c 923d4d7e 6d7c61b2
29e9c5a2 7eced3d9)

Appendix F. Further Mappings

The non-binary curves specified in Appendix A are expressed in different curve models, viz. as curves in short-Weierstrass form, as Montgomery curves, or as twisted Edwards curves. In Appendix D we already described relationships between these various curve models. Further mappings exist between elliptic curves within the same curve model. These can be exploited to force some of the domain parameters to specific values that allow for a more efficient implementation of the addition formulae.

F.1. Isomorphic Mapping between Twisted Edwards Curves

One can map points of the twisted Edwards curve E_{a,d} to points of the twisted Edwards curve E_{a',d'}, where a:=a'*s^2 and d:=d'*s^2 for some nonzero element s of GF(q). This defines a one-to-one

correspondence, which - in fact - is an isomorphism between E_{a,d} and E_{a',d'}.

The mapping from E_{a,d} to E_{a',d'} is defined by mapping the point (x,y) of E_{a,d} to the point (x', y'):=(s*x, y) of E_{a',d'}. The inverse mapping from E_{a',d'} to E_{a,d} is defined by mapping the point (x', y') of E_{a',d'} to the point (x, y):=(x'/s, y') of E_{a,d}.

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a twisted Edwards curve with generic domain parameters a and d on a corresponding isomorphic twisted Edwards curve with domain parameters a' and d' that have a more special form and that are known to allow for more efficient implementations of addition laws and translating the result back to the original curve. In particular, it is known that such efficiency improvements exist if a':=(+/-)1 (see [tEd-Formulas]).

F.2. Isomorphic Mapping between Montgomery Curves

One can map points of the Montgomery curve M_{A,B} to points of the Montgomery curve M_{A',B'}, where A:=A' and B:=B'*s^2 for some nonzero element s of GF(q). This defines a one-to-one correspondence, which - in fact - is an isomorphism between M_{A,B} and M_{A',B'}.

The mapping from M_{A,B} to M_{A',B'} is defined by mapping the point at infinity O of M_{A,B} to the point at infinity O of M_{A',B'}, while mapping each other point (u,v) of M_{A,B} to the point (u', v'):=(u, s*v) of M_{A',B'}. The inverse mapping from M_{A',B'} to M_{A,B} is defined by mapping the point at infinity O of M_{A',B'} to the point at infinity O of M_{A,B}, while mapping each other point (u',v') of M_{A',B'} to the point (u,v):=(u',v'/s) of M_{A,B}.

One can also map points of the Montgomery curve M_{A,B} to points of the Montgomery curve M_{A',B'}, where A':=-A and B':=-B. This defines a one-to-one correspondence, which - in fact - is an isomorphism between M_{A,B} and M_{A',B'}.

In this case, the mapping from M_{A,B} to M_{A',B'} is defined by mapping the point at infinity O of M_{A,B} to the point at infinity O of M_{A',B'}, while mapping each other point (u,v) of M_{A,B} to the point (u',v'):=(-u,v) of M_{A',B'}. The inverse mapping from M_{A',B'} to M_{A,B} is defined by mapping the point at infinity O of M_{A',B'} to the point at infinity O of M_{A,B}, while mapping each other point (u',v') of M_{A',B'} to the point (u,v):=(-u',v') of M_{A,B}.

Implementations may take advantage of these mappings to carry out elliptic curve groups operations originally defined for a Montgomery curve with generic domain parameters A and B on a corresponding isomorphic Montgomery curve with domain parameters A' and B' that have a more special form and that are known to allow for more efficient implementations of addition laws and translating the result back to the original curve. In particular, it is known that such efficiency improvements exist if B' assumes a small absolute value, such as B':=(+/-)1. (see [Mont-Ladder]).

F.3. Isomorphic Mapping between Weierstrass Curves

One can map points of the Weierstrass curve W_{a,b} to points of the Weierstrass curve W_{a',b'}, where a':=a*s^4 and b':=b*s^6 for some nonzero element s of GF(q). This defines a one-to-one correspondence, which - in fact - is an isomorphism between W_{a,b} and W_{a',b'}.

The mapping from W_{a,b} to W_{a',b'} is defined by mapping the point at infinity O of W_{a,b} to the point at infinity O of W_{a',b'}, while mapping each other point (X,Y) of W_{a,b} to the point (X',Y'):=(X*s^2, Y*s^3) of W_{a',b'}. The inverse mapping from W_{a',b'} to W_{a,b} is defined by mapping the point at infinity O of W_{a',b'} to the point at infinity O of W_{a,b}, while mapping each other point (X', Y') of W_{a',b'} to the point (X,Y):=(X'/s^2,Y'/s^3) of W_{a,b}.

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a Weierstrass curve with generic domain parameters a and b on a corresponding isomorphic Weierstrass curve with domain parameter a' and b' that have a more special form and that are known to allow for more efficient implementations of addition laws and translating the result back to the original curve. In particular, it is known that such efficiency improvements exist if a'=-3 (mod p), where p is the characteristic of GF(q), and one uses so-called Jacobian coordinates with a particular projective version of the addition laws of Appendix C.1. While not all Weierstrass curves can be put into this form, all traditional NIST curves have domain parameter a=-3, while all Brainpool curves [RFC5639] are isomorphic to a Weierstrass curve of this form via the above mapping.

Note that implementations for elliptic curves with short-Weierstrass form that hard-code the domain parameter a to a= -3 cannot always be used this way, since the curve W_{a,b} cannot always be expressed in terms of a Weierstrass curve with a'=-3 via a coordinate transformation: this only holds if a'/a is a fourth power in GF(q) (see Section 3.1.5 of [GECC]). However, even in this case, one can

still express the curve W_{a,b} as a Weierstrass curve with a small domain parameter value a', thereby still allowing a more efficient implementation than with a general domain parameter value a.

F.4. Isogenous Mapping between Weierstrass Curves

One can still map points of the Weierstrass curve W_{a,b} to points of the Weierstrass curve W_{a',b'}, where a':=-3 (mod p) and where p is the characteristic of GF(q), even if a'/a is not a fourth power in GF(q). In that case, this mappping cannot be an isomorphism (see Appendix F.3). Instead, the mapping is a so-called isogeny (or homomorphism). Since most elliptic curve operations process points of prime order or use so-called "co-factor multiplication", in practice the resulting mapping has similar properties as an isomorphism. In particular, one can still take advantage of this mapping to carry out elliptic curve group operations originally defined for a Weierstrass curve with domain parameter a unequal to -3 (mod p) on a corresponding isogenous Weierstrass curve with domain parameter a'=-3 (mod p) and translating the result back to the original curve.

In this case, the mapping from W_{a,b} to W_{a',b'} is defined by mapping the point at infinity O of W_{a,b} to the point at infinity O of W_{a',b'}, while mapping each other point (X,Y) of W_{a,b} to the point (X',Y'):=(u(X)/w(X)^2,Y*v(X)/w(X)^3) of W_{a',b'}. Here, u(X), v(X), and w(X) are polynomials in X that depend on the isogeny in question, as do domain parameters a' and b'. The inverse mapping from W_{a',b'} to W_{a,b} is again an isogeny (called the dual isogeny) and defined by mapping the point at infinity O of W_{a',b'} to the point at infinity O of W_{a,b}, while mapping each other point (X', Y') of W_{a',b'} to the point (X,Y):=(u'(X')/w'(X')^2,Y'*v'(X')/w'(X')^3) of W_{a,b}, where -- again -- u'(X'), v'(X'), and w'(X') are polynomials in X' that depend on the isogeny in question. These mappings have the property that their composition is not the identity mapping (as was the case with the isomorphic mappings discussed in Appendix F.3), but rather a fixed multiple hereof: if this multiple is l then the isogeny is called an isogeny of degree l (or l-isogeny) and u, v, and w (and, similarly, u', v', and w') are polynomials of degrees l, 3*(l-1)/2, and (l-1)/2, respectively. Note that an isomorphism is simply an isogeny of degree l=1. Details of how to determine isogenies are out of the scope of this document. The above formulas assume that the isogeny has odd degree (i.e., l is odd); detailed formulas for even-degree isogenies are similar, but out of scope.

Implementations may take advantage of this mapping to carry out elliptic curve group operations originally defined for a Weierstrass curve with a generic domain parameter a on a corresponding isogenous

Weierstrass curve with domain parameter a'=-3 (mod p), where one can use so-called Jacobian coordinates with a particular projective version of the addition laws of Appendix C.1. Since all traditional NIST curves have domain parameter a=-3, while all Brainpool curves [RFC5639] are isomorphic to a Weierstrass curve of this form, this allows taking advantage of existing implementations for these curves that may have a hardcoded a=-3 (mod p) domain parameter, provided one switches back and forth to this curve form using the isogenous mapping in question.

Note that isogenous mappings can be easily realized using representations in projective coordinates and involves roughly 3*l finite field multiplications, thus allowing switching between alternative representations at relatively low incremental cost compared to that of elliptic curve scalar multiplications (provided the isogeny has low degree l). Note, however, that this does require storage of the polynomial coefficients of the isogeny and dual isogeny involved. This illustrates that low-degree isogenies are to be preferred, since an l-isogeny (usually) requires storing roughly 6*l elements of GF(q). While there are many isogenies, we therefore only consider those with the desired property with lowest possible degree.

Appendix G. Further Cousins of Curve25519

This section introduces some further curves related to Curve25519 and explains their relationships.

G.1. Further Alternative Representations

The Weierstrass curve Wei25519 is isomorphic to the Weierstrass curve Wei25519.2 defined over GF(p), with as base point the pair (G2X,G2Y), and isogenous to the Weierstrass curve Wei25519.-3 defined over GF(p), with as base point the pair (G3X, G3Y), where parameters are as specified in Appendix G.3 and where the related mappings are as specified in Appendix G.2.

G.2. Further Switching

Each affine point (X, Y) of Wei25519 corresponds to the point (X', Y'):=(X*s^2,Y*s^3) of Wei25519.2, where s is the element of GF(p) defined by

s 20343593038935618591794247374137143598394058341193943326473831977
39407761440

(=0x047f6814 6d568b44 7e4552ea a5ed633d 02d62964 a2b0a120
5e7941e9 375de020),

while the point at infinity of Wei25519 corresponds to the point at infinity of Wei25519.2. (Here, we used the mapping of Appendix F.3.) Under this mapping, the base point (GX, GY) of Wei25519 corresponds to the base point (G2X,G2Y) of Wei25519.2. The inverse mapping maps the affine point (X', Y') of Wei25519.2 to (X,Y):=(X'/s^2,Y'/s^3) of Wei25519, while mapping the point at infinity O of Wei25519.2 to the point at infinity O of Wei25519. Note that this mapping (and its inverse) involves a modular multiplication of both coordinates with fixed constants s^2 and s^3 (respectively, 1/s^2 and 1/s^3), which can be precomputed.

Each affine point (X,Y) of Wei25519 corresponds to the point (X',Y'):=(X1*t^2,Y1*t^3) of Wei25519.-3, where (X1,Y1)=(u(X)/w(X)^2,Y*v(X)/w(X)^3), where u, v, and w are the polynomials with coefficients in GF(p) as defined in Appendix G.4.1 and where t is the element of GF(p) defined by

t 357281333982891756495869386056605426886916156991696629671545250846441815962 29

(=0x4efd6829 88ff8526 e189f712 5999550c e9ef729b ed1a7015 73b1bab8 8bfcd845),

while the point at infinity of Wei25519 corresponds to the point at infinity of Wei25519.-3. (Here, we used the isogenous mapping of Appendix F.4.) Under this isogenous mapping, the base point (GX,GY) of Wei25519 corresponds to the base point (G3X,G3Y) of Wei25519.-3. The dual isogeny maps the affine point (X',Y') of Wei25519.-3 to the affine point (X,Y):=(u'(X1)/w'(X1)^2,Y1*v'(X1)/w'(X1)^3) of Wei25519, where (X1,Y1)=(X'/t^2,Y'/t^3) and where u', v', and w' are the polynomials with coefficients in GF(p) as defined in Appendix G.4.2, while mapping the point at infinity O of Wei25519.-3 to the point at infinity O of Wei25519. Under this dual isogenous mapping, the base point (G3X, G3Y) of Wei25519.-3 corresponds to a multiple of the base point (GX, GY) of Wei25519, where this multiple is l=47 (the degree of the isogeny; see the description in Appendix F.4). Note that this isogenous map (and its dual) primarily involves the evaluation of three fixed polynomials involving the x-coordinate, which takes roughly 140 modular multiplications (or less than 5-10% relative incremental cost compared to the cost of an elliptic curve scalar multiplication).

G.3. Further Domain Parameters

The parameters of the Weierstrass curve with a=2 that is isomorphic with Wei25519 and the parameters of the Weierstrass curve with a=-3 that is isogenous with Wei25519 are as indicated below. Both domain

parameter sets can be exploited directly to derive more efficient point addition formulae, should an implementation facilitate this.

General parameters: same as for Wei25519 (see Appendix E.3)

Weierstrass curve-specific parameters (for Wei25519.2, i.e., with a=2):

a 2 (=0x02)

b 12102640281269758552371076649779977768474709596484288167752775713178787220689

(=0x1ac1da05 b55bc146 33bd39e4 7f94302e f19843dc f669916f 6a5dfd01 65538cd1)

G2X 10770553138368400518417020196796161136792368198326337823149502681097436401658

(=0x17cfeac3 78aed661 318e8634 582275b6 d9ad4def 072ea193 5ee3c4e8 7a940ffa)

G2Y 54430575861508405653098668984457528616807103332502577521161439773 88639873869

(=0x0c08a952 c55dfad6 2c4f13f1 a8f68dca dc5c331d 297a37b6 f0d7fdcc 51e16b4d)

Weierstrass curve-specific parameters (for Wei25519.-3, i.e., with a=-3):

a -3

(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffea)

b 29689592517550930188872794512874050362622433571298029721775200646451501277098

(=0x41a3b6bf c668778e be2954a4 b1df36d1 485ecef1 ea614295 796e1022 40891faa)

G3X 53837179229940872434942723257480777370451127212339198133697207846219400243292

(=0x7706c37b 5a84128a 3884a5d7 1811f1b5 5da3230f fb17a8ab 0b32e48d 31a6685c)

G3Y 6954807309110018441440205552927997039251486742285514177307080418460388229929

(=0x0f60480c 7a5c0e11 40340adc 79d6a2bf 0cb57ad0 49d025dc 38d80c77 985f0329)

G.4. Isogeny Details

The isogeny and dual isogeny are both isogenies with degree l=47. Both are specified by a triple of polynomials u, v, and w (resp. u', v', and w') of degree 47, 69, and 23, respectively, with coefficients in GF(p). The coeffients of each of these polynomials are specified in Appendix G.4.1 (for the isogeny) and in Appendix G.4.2 (for the dual isogeny). For each polynomial in variable x, the coefficients are tabulated as the sequence of coefficients of x^0, x^1, x^2, ..., in hexadecimal format.

G.4.1. Isogeny Parameters

G.4.1.1. Coefficients of u(x)

0 0x670ed14828b6f1791ceb3a9cc0edfe127dee8729c5a72ddf77bb1abaebbba1e8

1 0x1135ca8bd5383cb3545402c8bce2ced14b45c29b241e4751b035f27524a9f932

2 0x3223806ff5f669c430efd74df8389f058d180e2fcffa5cdef3eacecdd2c34771

3 0x31b8fecf3f17a819c228517f6cd9814466c8c8bea2efccc47a29bfc14c364266

4 0x2541305c958c5a326f44efad2bec284e7abee840fadb08f2d994cd382fd8ce42

5 0x6e6f9c5792f3ff497f860f44a9c469cec42bd711526b733e10915be5b2dbd8c6

6 0x3e9ad2e5f594b9ce6b06d4565891d28a1be8790000b396ef0bf59215d6cabfde

7 0x278448895d236403bbc161347d19c913e7df5f372732a823ed807ee1d30206be

8 0x42f9d171ea8dc2f4a14ea46cc0ee54967175ecfe83a975137b753cb127c35060

9 0x128e40efa2d3ccb51567e73bae91e7c31eac45700fa13ce5781cbe5ddc985648

10 0x450e5086c065430b496d88952dd2d5f2c5102bc27074d4d1e98bfa47413e0645

11 0x487ef93da70dfd44a4db8cb41542e33d1aa32237bdca3a59b3ce1c59585f253d

12 0x33d209270026b1d2db96efb36cc2fa0a49be1307f49689022eab1892b010b785

13 0x4732b5996a20ebc4d5c5e2375d3b6c4b700c681bd9904343a14a0555ef0ecd48

```
   14 0x64dc9e8272b9f5c6ad3470db543238386f42b18cb1c592cc6caf7893141b2107
   15 0x52bbacd1f85c61ef7eafd8da27260fa2821f7a961867ed449b283036508ac5c5
   16 0x320447ed91210985e2c401cfe1a93db1379424cf748f92fd61ab5cc356bc89a2
   17 0x23d23a49bbcdf8cf4c4ce8a4ff7dd87d1ad1970317686254d5b4d2ec050d019f
   18 0x1601fca063f0bbbf15f198b3c20e474c2170294fa981f73365732d2372b40cd4
   19 0x7bf3f93840035e9688cfff402cee204a17c0de9779fc33503537dd78021bf4c4
   20 0x311998ce59fb7e1cd6af591ece3e84dfcb1c330cbcf28c0349e37b9581452853
   21 0x7ae5e41acfd28a9add2216dfed34756575a19b16984c1f3847b694326dad7f99
   22 0x704957e279244a5b107a6c57bd0ab9afe5227b7c0be2052cd3513772a40efee7
   23 0x56b918b5a0c583cb763550f8f71481e57c13bdcef2e5cfc8091d0821266f233b
   24 0x677073fed43ab291e496f798fbcf217bac3f014e35d0c2fa07f041ae746a04d7
   25 0x22225388e76f9688c7d4053b50ba41d0d8b71a2f21da8353d98472243ef50170
   26 0x66930b3dffdd3995a2502cef790d78b091c875192d8074bb5d5639f736400555
   27 0x79eb677c5e36971e8d64d56ebc0dedb4e9b7dd2d7b01343ebbd4d358d376e490
   28 0x48a204c2ca6d8636e9994842605bd648b91b637844e38d6c7dd707edce8256e2
   29 0x0fb3529b0d4b9ce2d70760f33e8ce997a58999718e9277caf48623d27ae6a788
   30 0x4352604bffd0c7d7a9ed898a2c6e7cf2512ffb89407271ba1f2c2d0ead8cc5aa
   31 0x6667697b29785fb6f0bd5e04d828991a5fe525370216f347ec767a26e7aac936
   32 0x09fc950b083c56dbd989badf9887255e203c879f123a7cb28901e50aea6d64dc
   33 0x41e51b51b5caadd1c15436bbf37596a1d7288a5f495d6b5b1ae66f8b2942b31d
   34 0x073b59fec709aa1cabd429e981c6284822a8b7b07620c831ab41fd31d5cf7430
   35 0x67e9b88e9a1bfbc2554107d67d814986f1b09c3107a060cba21c019a2d5dc848
   36 0x6881494a1066ca176c5e174713786040affb4268b19d2abf28ef4293429f89c1
   37 0x5f4d30502ff1e1ccd624e6f506569454ab771869d7483e26afc09dea0c5ccd3d
```

38 0x02a814cfc5859bca51e539c159955cbe729a58978b52329575d09bc6c3bf97ad

39 0x1313c8aaae20d6f4397f0d8b19e52cfcdf8d8e10fba144aec1778fd10ddf4e9c

40 0x7008d38f434b98953a996d4cc79fcbef9502411dcdf92005f725cea7ce82ad47

41 0x5a74d1296aaaa245ffb848f434531fa3ba9e5cb9098a7091d36c2777d4cf5a13

42 0x4bd3b700606397083f8038177bdaa1ac6edbba0447537582723cae0fd29341a9

43 0x573453fb2b093016f3368356c786519d54ed05f5372c01723b4da520597ec217

44 0x77f5c605bdb3a30d7d9c8840fce38650910d4418eed707a212c8927f41c2c812

45 0x16d6b9f7ff57ca32350057de1204cc6d69d4ef1b255dfef8080118e2fef6ace3

46 0x34e8595832a4021f8b5744014c6b4f7da7df0d0329e8b6b4d44c8fadad6513b7

47 0x01

G.4.1.2. Coefficients of v(x)

0 0x0f9f5eb7134e6f8dafa30c45afa58d7bfc6d4e3ccbb5de87b562fd77403972b2

1 0x36c2dcd9e88f0d2d517a15fc453a098bbbb5a05eb6e8da906fae418a4e1a13f7

2 0x0b40078302c24fa394a834880d5bf46732ca1b4894172fb7f775821276f558b3

3 0x53dd8e2234573f7f3f7df11e90a7bdd7b75d807f9712f521d4fb18af59aa5f26

4 0x6d4d7bb08de9061988a8cf6ff3beb10e933d4d2fbb8872d256a38c74c8c2ceda

5 0x71bfe5831b30e28cd0fbe1e9916ab2291c6beacc5af08e2c9165c632e61dd2f5

6 0x7c524f4d17ff2ee88463da012fc12a5b67d7fb5bd0ab59f4bbf162d76be1c89c

7 0x758183d5e07878d3364e3fd4c863a5dc1fe723f48c4ab4273fc034f5454d59a4

8 0x1eb41ef2479444ecdccbc200f64bde53f434a02b6c3f485d32f14da6aa7700e1

9 0x1490f3851f016cc3cf8a1e3c16a53317253d232ed425297531b560d70770315c

10 0x09bc43131964e46d905c3489c9d465c3abbd26eab9371c10e429b36d4b86469c

11 0x5f27c173d94c7a413a288348d3fc88daa0bcf5af8f436a47262050f240e9be3b

12 0x1d20010ec741aaa393cd19f0133b35f067adab0d105babe75fe45c8ba2732ceb

```
   13 0x01b3c669ae49b86be2f0c946a9ff6c48e44740d7d9804146915747c3c025996a
   14 0x24c6090f79ec13e3ae454d8f0f98e0c30a8938180595f79602f2ba013b3c10db
   15 0x4650c5b5648c6c43ac75a2042048c699e44437929268661726e7182a31b1532f
   16 0x0957a835fb8bac3360b5008790e4c1f3389589ba74c8e8bf648b856ba7f22ba5
   17 0x1cd1300bc534880f95c7885d8df04a82bd54ed3e904b0749e0e3f8cb3240c7c7
   18 0x760b486e0d3c6ee0833b34b64b7ebc846055d4d1e0beeb6aedd5132399ada0ea
   19 0x1c666846c63965ef7edf519d6ada738f2b676ae38ff1f4621533373931b3220e
   20 0x365055118b38d4bc0df86648044affea2ef33e9a392ad336444e7d15e45585d1
   21 0x736487bde4b555abfccd3ea7ddcda98eda0d7c879664117dee906a88bc551194
   22 0x70de05ab9520222a37c7a84c61eedff71cb50c5f6647fc2a5d6e0ff2305cea37
   23 0x59053f6cdf6517ab3fe4bd9c9271d1892f8cf353d8041b98409e1e341a01f8b5
   24 0x375db54ed12fe8df9a198ea40200e812c2660b7022681d7932d89fafe7c6e88d
   25 0x2a070c31d1c1a064daf56c79a044bd1cd6d13f1ddb0ff039b03a6469aaa9ed77
   26 0x41482351e7f69a756a5a2c0b3fa0681c03c550341d0ca0f76c5b394db9d2de8d
   27 0x747ac1109c9e9368d94a302cb5a1d23fcc7f0fd8a574efb7ddcaa738297c407a
   28 0x45682f1f2aab6358247e364834e2181ad0448bb815c587675fb2fee5a2119064
   29 0x148c5bf44870dfd307317f0a0e4a8c163940bee1d2f01455a2e658aa92c13620
   30 0x6add1361e56ffa2d2fbbddba284b35be5845aec8069fc28af009d53290a705ce
   31 0x6631614c617400dc00f2c55357f67a94268e7b5369b02e55d5db46c935be3af5
   32 0x17cffb496c64bb89d91c8c082f4c288c3c87feabd6b08591fe5a92216c094637
   33 0x648ff88155969f54c955a1834ad227b93062bb191170dd8c4d759f79ad5da250
   34 0x73e50900b89e5f295052b97f9d0c9edb0fc7d97b7fa5e3cfeefe33dd6a9cb223
   35 0x6afcb2f2ffe6c08508477aa4956cbd3dc864257f5059685adf2c68d4f2338f00
   36 0x372fd49701954c1b8f00926a8cb4b157d4165b75d53fa0476716554bf101b74c
```

```
   37 0x0334ed41325f3724ff8becbf2b3443fea6d30fa543d1ca13188aceb2bdaf5f4e
   38 0x70e629c95a94e8e1b3974acb25e18ba42f8d5991786f0931f650c283adfe82fd
   39 0x738a625f4c62d3d645f1274e09ab344e72d441f3c0e82989d3e21e19212f23f3
   40 0x7093737294b29f21522f5664a9941c9b476f75d443b647bd2c777040bcd12a6a
   41 0x0a996bad5863d821ccb8b89fa329ddbe5317a46bcb32552db396bea933765436
   42 0x2da237e3741b75dd0264836e7ef634fc0bc36ab187ebc790591a77c257b06f53
   43 0x1902f3daa86fa4f430b57212924fdc9e40f09e809f3991a0b3a10ab186c50ee5
   44 0x12baffec1bf20c921afd3cdf67a7f1d87c00d5326a3e5c83841593c214dadcb1
   45 0x6460f5a68123cb9e7bc1289cd5023c0c9ccd2d98eea24484fb3825b59dcd09aa
   46 0x2c7d63a868ffc9f0fd034f821d84736c5bc33325ce98aba5f0d95fef6f230ec8
   47 0x756e0063349a702db7406984c285a9b6bfba48177950d4361d8efa77408dc860
   48 0x037f3e30032b21e0279738e0a2b689625447831a2ccf15c638672da9aa7255ae
   49 0x1107c0dbe15d6ca9e790768317a40bcf23c80f1841f03ca79dd3e3ef4ea1ae30
   50 0x61ff7f25721d6206041c59a788316b09e05135a2aad94d539c65daa68b302cc2
   51 0x5dbfe346cbd0d61b9a3b5c42ec0518d3ae81cabcc32245060d7b0cd982b8d071
   52 0x4b6595e8501e9ec3e75f46107d2fd76511764efca179f69196eb45c0aa6fade3
   53 0x72d17a5aa7bd8a2540aa9b02d9605f2a714f44abfb4c35d518b7abc39b477870
   54 0x658d8c134bac37729ec40d27d50b637201abbf1ab4157316358953548c49cf22
   55 0x36ac53b9118581ace574d5a08f9647e6a916f92dda684a4dbc405e2646b0243f
   56 0x1917a98f387d1e323e84a0f02d53307b1dd949e1a27b0de14514f89d9c0ef4b6
   57 0x21573434fde7ce56e8777c79539479441942dba535ade8ecb77763f7eb05d797
   58 0x0e0bf482dc40884719bea5503422b603f3a8edb582f52838caa6eaab6eeac7ef
   59 0x3b0471eb53bd83e14fbc13928fe1691820349a963be8f7e9815848a53d03f5eb
   60 0x1e92cb067b24a729c42d3abb7a1179c577970f0ab3e6b0ce8d66c5b8f7001262
```

61 0x74ea885c1ebed6f74964262402432ef184c42884fceb2f8dba3a9d67a1344dd7

62 0x433ebce2ce9b0dc314425cfc2b234614d3c34f2c9da9fff4fdddd1ce242d035b

63 0x33ac69e6be858dde7b83a9ff6f11de443128b39cec6e410e8d3b570e405ff896

64 0x0dab71e2ae94e6530a501ed8cf3df26731dd1d41cd81578341e12dca3cb71aa3

65 0x537f58d52d18ce5b1d5a6bd3a420e796e64173491ad43dd4d1083a7dcc7dd201

66 0x49c2f6afa93fdcc4e0f8128a8b06da4c75049be14edf3e103821ab604c60f8ae

67 0x10a333eabd6135aeaa3f5f5f7e73d102e4fd7e4bf0902fc55b00da235fa1ad08

68 0x0f5c86044bf6032f5102e601f2a0f73c7bce9384bedd120f3e72d78484179d9c

69 0x01

G.4.1.3. Coefficients of w(x)

0 0x3da24d42421264f30939ff00203880f2b017eb3fecf8933ae61e18df8c8ba116

1 0x0457f20bc393cdc9a66848ce174e2fa41d77e6dbae05a317a1fb6e3ae78760f8

2 0x7f608a2285c480d5c9592c435431fae94695beef79d770bb6d029c1d10a53295

3 0x3832accc520a485100a0a1695792465142a5572bed1b2e50e1f8f662ac7289bb

4 0x2df1b0559e31b328eb34beedd5e537c3f4d7b9befb0749f75d6d0d866d26fbaa

5 0x25396820381d04015a9f655ddd41c74303ded05d54a7750e2f58006659adda28

6 0x6fa070a70ca2bc6d4d0795fb28d4990b2cc80cd72d48b603a8ac8c8268bef6a6

7 0x27f488578357388b20fbc7503328e1d10de602b082b3c7b8ceb33c29fea7a0d2

8 0x15776851a7cabcfe84c632118306915c0c15c75068a47021968c7438d46076e6

9 0x101565b08a9af015c172fb194b940a4df25c4fb1d85f72d153efc79131d45e8f

10 0x196b0ffbf92f3229fea1dac0d74591b905ccaab6b83f905ee813ee8449f8a62c

11 0x01f55784691719f765f04ee9051ec95d5deb42ae45405a9d87833855a6d95a94

12 0x628858f79cca86305739d084d365d5a9e56e51a4485d253ae3f2e4a379fa8aff

13 0x4a842dcd943a80d1e6e1dab3622a8c4d390da1592d1e56d1c14c4d3f72dd01a5

14 0x0f3bfc9cb17a1125f94766a4097d0f1018963bc11cb7bc0c7a1d94d65e282477

15 0x1c4bd70488c4882846500691fa7543b7ef694446d9c3e3b4707ea2c99383e53c

16 0x2d7017e47b24b89b0528932c4ade43f09091b91db0072e6ebdc5e777cb215e35

17 0x781d69243b6c86f59416f91f7decaca93eab9cdc36a184191810c56ed85e0fdc

18 0x5f20526f4177357da40a18da054731d442ad2a5a4727322ba8ed10d32eca24fb

19 0x33e4cab64ed8a00d8012104fe8f928e6173c428eff95bbbe569ea46126a4f3cd

20 0x050555b6f07e308d33776922b6566829d122e19b25b7bbacbb0a4b1a7dc40192

21 0x533fa4bf1e2a2aae2f979065fdbb5b667ede2f85543fddbba146aa3a4ef2d281

22 0x5a742cac1952010fc5aba200a635a7bed3ef868194f45b5a6a2647d6d6b289d2

23 0x01

G.4.2. Dual Isogeny Parameters

G.4.2.1. Coefficients of u'(x)

0 0x0f0eddb584a20aaac8f1419efdd02a5cca77b21e4cfae78c49b5127d98bc5882

1 0x7115e60d44a58630417df33dd45b8a546fa00b79fea3b2bdc449694bade87c0a

2 0x0b3f3a6f3c445c7dc1f91121275414e88c32ff3f367ba0edad4d75b7e7b94b65

3 0x1eb31bb333d7048b87f2b3d4ec76d69035927b41c30274368649c87c52e1ab30

4 0x552c886c2044153e280832264066cce2a7da1127dc9720e2a380e9d37049ac64

5 0x4504f27908db2e1f5840b74ae42445298755d9493141f5417c02f04d47797dda

6 0x082c242cce1eb19698a4fa30b5affe64e5051c04ae8b52cb68d89ee85222e628

7 0x480473406add76cf1d77661b3ff506c038d9cdd5ad6e1ea41969430bb876d223

8 0x25f47bb506fba80c79d1763365fa9076d4c4cb6644f73ed37918074397e88588

9 0x10f13ed36eab593fa20817f6bb70cac292e18d300498f6642e35cbdf772f0855

10 0x7d28329d695fb3305620f83a58df1531e89a43c7b3151d16f3b60a8246c36ade

11 0x02c5ec8c42b16dc6409bdd2c7b4ffe9d65d7209e886badbd5f865dec35e4ab4a

```
   12 0x7f4f33cd50255537e6cde15a4a327a5790c37e081802654b56c956434354e133
   13 0x7d30431a121d9240c761998cf83d228237e80c3ef5c7191ec9617208e0ab8cec
   14 0x4d2a7d6609610c1deed56425a4615b92f70a507e1079b2681d96a2b874cf0630
   15 0x74676df60a9906901d1dc316c639ff6ae0fcdb02b5571d4b83fc2eedcd2936a8
   16 0x22f8212219aca01410f06eb234ed53bd5b8fbe7c08652b8002bcd1ea3cdae387
   17 0x7edb04449565d7c566b934a87fadade5515f23bda1ce25daa19fff0c6a5ccc2f
   18 0x106ef71aa3aa34e8ecf4c07a67d03f0949d7d015ef2c1e32eb698dd3bec5a18c
   19 0x0017913eb705db126ac3172447bcd811a62744d505ad0eea94cfcfdde5ca7428
   20 0x2cc793e6d3b592dcf5472057a991ff1a5ab43b4680bb34c0f5faffc5307827c1
   21 0x6dafcc0b16f98300cddb5e0a7d7ff04a0e73ca558c54461781d5a5ccb1ea0122
   22 0x7e418891cf222c021b0ae5f5232b9c0dc8270d4925a13174a0f0ac5e7a4c8045
   23 0x76553bd26fecb019ead31142684789fea7754c2dc9ab9197c623f45d60749058
   24 0x693efb3f81086043656d81840902b6f3a9a4b0e8f2a5a5edf5ce1c7f50a3898e
   25 0x46c630eac2b86d36f18a061882b756917718a359f44752a5caf41be506788921
   26 0x01dcfa01773628753bc6f448ac11be8a3bffa0011b9284967629b827e064f614
   27 0x08430b5b97d49b0938d1f66ecb9d2043025c6eec624f8f02042b9621b2b5cb19
   28 0x66f66a6669272d47d3ec1efea36ee01d4a54ed50e9ec84475f668a5a9850f9be
   29 0x539128823b5ef3e87e901ab22f06d518a9bad15f5d375b49fe1e893ab38b1345
   30 0x2bd01c49d6fff22c213a8688924c10bf29269388a69a08d7f326695b3c213931
   31 0x3f7bea1baeccea3980201dc40d67c26db0e3b15b5a19b6cdac6de477aa717ac1
   32 0x6e0a72d94867807f7150fcb1233062f911b46e2ad11a3eac3c6c4c91e0f4a3fa
   33 0x5963f3cc262253f56fc103e50217e7e5b823ae8e1617f9e11f4c9c595fbb5bf6
   34 0x41440b6fe787777bc7b63afac9f4a38ddadcebc3d72f8fc73835247ba05f3a1d
   35 0x66d185401c1d2d0b84fcf6758a6a985bf9695651271c08f4b69ce89175fb7b34
```

36 0x2673fb8c65bc4fe41905381093429a2601c46a309c03077ca229bac7d6ccf239

37 0x1ce4d895ee601918a080de353633c82b75a3f61e8247763767d146554dd2f862

38 0x18efa6c72fa908347547a89028a44f79f22542baa588601f2b3ed25a5e56d27c

39 0x53de362e2f8ff220f8921620a71e8faa1aa57f8886fcbb6808fa3a5560570543

40 0x0dc29a73b97f08aa8774911474e651130ed364e8d8cffd4a80dee633aacecc47

41 0x4e7eb8584ae4de525389d1e9300fc4480b3d9c8a5a45ecfbe33311029d8f6b99

42 0x6c3cba4aa9229550fa82e1cfaee4b02f2c0cb86f79e0d412b8e32b00b7959d80

43 0x5a9d104ae585b94af68eeb16b1349776b601f97b7ce716701645b1a75b68dcf3

44 0x754e014b5e87af035b3d5fe6fb49f4631e32549f6341c6693c5172a6388e273e

45 0x6710d8265118e22eaceba09566c86f642ab42da58c435083a353eaa12d866c39

46 0x6e88ac659ce146c369f8b24c3a49f8dca547827250cf7963a455851cfc4f8d22

47 0x0971eb5f253356cd1fde9fb21f4a4902aa5b8d804a2b57ba775dc130181ae2e8

G.4.2.2. Coefficients of v'(x)

0 0x043c9b67cc5b16e167b55f190db61e44d48d813a7112910f10e3fd8da85d61d3

1 0x72046db07e0e7882ff3f0f38b54b45ca84153be47a7fd1dd8f6402e17c47966f

2 0x1593d97b65a070b6b3f879fe3dc4d1ef03c0e781c997111d5c1748f956f1ffc0

3 0x54e5fec076b8779338432bdc5a449e36823a0a7c905fd37f232330b026a143a0

4 0x46328dd9bc336e0873abd453db472468393333fbf2010c6ac283933216e98038

5 0x25d0c64de1dfe1c6d5f5f2d98ab637d8b39bcf0d886a23dabac18c80d7eb03ce

6 0x3a175c46b2cd8e2b313dde2d5f3097b78114a6295f283cf58a33844b0c8d8b34

7 0x5cf4e6f745bdd61181a7d1b4db31dc4c30c84957f63cdf163bee5e466a7a8d38

8 0x639071c39b723eea51cfd870478331d60396b31f39a593ebdd9b1eb543875283

9 0x7ea8f895dcd85fc6cb2b58793789bd9246e62fa7a8c7116936876f4d8dff869b

10 0x503818acb535bcaacf8ad44a83c213a9ce83af7c937dc9b3e5b6efedc0a7428c

```
   11 0x0e815373920ec3cbf3f8cae20d4389d367dc4398e01691244af90edc3e6d42b8
   12 0x7e4b23e1e0b739087f77910cc635a92a3dc184a791400cbceae056c19c853815
   13 0x145322201db4b5ec0a643229e07c0ab7c36e4274745689be2c19cfa8a702129d
   14 0x0fde79514935d9b40f52e33429621a200acc092f6e5dec14b49e73f2f59c780d
   15 0x37517ac5c04dc48145a9d6e14803b8ce9cb6a5d01c6f0ad1b04ff3353d02d815
   16 0x58ae96b8eefe9e80f24d3b886932fe3c27aaea810fa189c702f93987c8c97854
   17 0x6f6402c90fa379096d5f436035bebc9d29302126e9b117887abfa7d4b3c5709a
   18 0x01dbdf2b9ec09a8defeb485cc16ea98d0d45c5b9877ff16bd04c0110d2f64961
   19 0x53c51706af523ab5b32291de6c6b1ee7c5cbd0a5b317218f917b12ff38421452
   20 0x1b1051c7aec7d37a349208e3950b679d14e39f979db4fcd7b50d7d27dc918650
   21 0x1547e8d36262d5434cfb029cdd29385353124c3c35b1423c6cca1f87910b305b
   22 0x198efe984efc817835e28f704d41e4583a1e2398f7ce14045c4575d0445c6ce7
   23 0x492276dfe9588ee5cd9f553d990f377935d721822ecd0333ce2eb1d4324d539c
   24 0x77bad5319bacd5ed99e1905ce2ae89294efa7ee1f74314e4095c618a4e580c9b
   25 0x2cb3d532b8eac41c61b683f7b02feb9c2761f8b4286a54c3c4b60dd8081a312e
   26 0x37d189ea60443e2fee9b7ba8a34ed79ff3883dcefc06592836d2a9dd2ee3656e
   27 0x79a80f9a0e6b8ded17a3d6ccf71eb565e3704c3543b77d70bca854345e880aba
   28 0x47718530ef8e8c75f069acb2d9925c5537908e220b28c8a2859b856f46d5f8db
   29 0x7dc518f82b55a36b4fa084b05bf21e3efce481d278a9f5c6a49701e56dac01ec
   30 0x340a318dad4b8d348a0838659672792a0f00b7105881e6080a340f708a9c7f94
   31 0x55f04d9d8891636d4e9c808a1fa95ad0dae7a8492257b20448023aad3203278e
   32 0x39dc465d58259f9f70bb430d27e2f0ab384a550e1259655443e14bdecba85530
   33 0x757385464cff265379a1adfadfd6f6a03fa8a2278761d4889ab097eff4d1ac28
   34 0x4d575654dbe39778857f4e688cc657416ce524d54864ebe8995ba766efa7ca2b
```

```
35 0x47adb6aecc1949f2dc9f01206cc23eb4a0c29585d475dd24dc463c5087809298
36 0x30d39e8b0c451a8fcf3d2abab4b86ffa374265abbe77c5903db4c1be8cec7672
37 0x28cf47b39112297f0daeaa621f8e777875adc26f35dec0ba475c2ee148562b41
38 0x36199723cc59867e2e309fe9941cd33722c807bb2d0a06eeb41de93f1b93f2f5
39 0x5cdeb1f2ee1c7d694bdd884cb1c5c22de206684e1cafb8d3adb9a33cb85e19a2
40 0x0f6e6b3fc54c2d25871011b1499bb0ef015c6d0da802ae7eccf1d8c3fb73856c
41 0x0c1422c98b672414344a9c05492b926f473f05033b9f85b8788b4bb9a080053c
42 0x19a8527de35d4faacb00184e0423962247319703a815eecf355f143c2c18f17f
43 0x7812dc3313e6cf093da4617f06062e8e8969d648dfe6b5c331bccd58eb428383
44 0x61e537180c84c79e1fd2d4f9d386e1c4f0442247605b8d8904d122ee7ef9f7be
45 0x544d8621d05540576cfc9b58a3dab19145332b88eb0b86f4c15567c37205adf9
46 0x11be3ef96e6e07556356b51e2479436d9966b7b083892b390caec22a117aa48e
47 0x205cda31289cf75ab0759c14c43cb30f7287969ea3dc0d5286a3853a4d403187
48 0x048d8fc6934f4f0a99f0f2cc59010389e2a0b20d6909bfcf8d7d0249f360acdc
49 0x42cecc6d9bdca6d382e97fcea46a79c3eda2853091a8f399a2252115bf9a1454
50 0x0117d41b24f2f69cb3270b359c181607931f62c56d070bbd14dc9e3f9ab1432e
51 0x7c51564c66f68e2ad4ce6ea0d68f920fafa375376709c606c88a0ed44207aa1e
52 0x48f25191fc8ac7d9f21adf6df23b76ccbca9cb02b815acdbebfa3f4eddc71b34
53 0x4fc21a62c4688de70e28ad3d5956633fc9833bc7be09dc7bc500b7fae1e1c9a8
54 0x1f23f25be0912173c3ef98e1c9990205a69d0bf2303d201d27a5499247f06789
55 0x3131495618a0ac4cb11a702f3f8bab66c4fa1066d0a741af3c92d5c246edd579
56 0x0d93fe40faa53913638e497328a1b47603cb062c7afc9e96278603f29fd11fd4
57 0x6b348bc59e984c91d696d1e3c3cfae44021f06f74798c787c355437fb696093d
58 0x65af00e73043edcb479620c8b48098b89809d577a4071c8e33e8678829138b8a
```

```
   59 0x5e62ffb032b2ddb06591f86a46a18effd5d6ecf3f129bb2bacfd51a3739a98b6
   60 0x62c974ef3593fc86f7d78883b8727a2f7359a282cbc0196948e7a793e60ce1a1
   61 0x204d708e3f500aad64283f753e7d9bab976aa42a4ca1ce5e9d2264639e8b1110
   62 0x0a90f0059da81a012e9d0a756809fab2ce61cb45965d4d1513a06227783ee4ea
   63 0x39fa55971c9e833f61139c39e243d40869fd7e8a1417ee4e7719dd2dd242766f
   64 0x22677c1e659caa324f0c74a013921facf62d0d78f273563145cc1ddccfcc4421
   65 0x3468cf6df7e93f7ff1fe1dd7e180a89dec3ed4f72843b4ea8a8d780011a245b2
   66 0x68f75a0e2210f52a90704ed5f511918d1f6bcfcd26b462cc4975252369db6e9d
   67 0x6220c0699696e9bcab0fe3a80d437519bd2bdf3caef665e106b2dd47585ddd9f
   68 0x553ad47b129fb347992b576479b0a89f8d71f1196f83e5eaab5f533a1dd6f6d7
   69 0x239aef387e116ec8730fa15af053485ca707650d9f8917a75f22acf6213197df
```

G.4.2.3. Coefficients of w'(x)

```
   0  0x6bd7f1fc5dd51b7d832848c180f019bcbdb101d4b3435230a79cc4f95c35e15e
   1  0x17413bb3ee505184a504e14419b8d7c8517a0d268f65b0d7f5b0ba68d6166dd0
   2  0x47f4471beed06e5e2b6d5569c20e30346bdba2921d9676603c58e55431572f90
   3  0x2af7eaafd04f6910a5b01cdb0c27dca09487f1cd1116b38db34563e7b0b414eb
   4  0x57f0a593459732eef11d2e2f7085bf9adf534879ba56f7afd17c4a40d3d3477b
   5  0x4da04e912f145c8d1e5957e0a9e44cca83e74345b38583b70840bdfdbd0288ed
   6  0x7cc9c3a51a3767d9d37c6652c349adc09bfe477d99f249a2a7bc803c1c5f39ed
   7  0x425d7e58b8adf87eebf445b424ba308ee7880228921651995a7eab548180ad49
   8  0x48156db5c99248234c09f43fedf509005943d3d5f5d7422621617467b06d314f
   9  0x0d837dbbd1af32d04e2699cb026399c1928472aa1a7f0a1d3afd24bc9923456a
   10 0x5b8806e0f924e67c1f207464a9d025758c078b43ddc0ea9afe9993641e5650be
   11 0x29c91284e5d14939a6c9bc848908bd9df1f8346c259bbd40f3ed65182f3a2f39
```

```
12 0x25550b0f3bceef18a6bf4a46c45bf1b92f22a76d456bfdf19d07398c80b0f946

13 0x495d289b1db16229d7d4630cb65d52500256547401f121a9b09fb8e82cf01953

14 0x718c8c610ea7048a370eabfd9888c633ee31dd70f8bcc58361962bb08619963e

15 0x55d8a5ceef588ab52a07fa6047d6045550a5c52c91cc8b6b82eeb033c8ca557d

16 0x620b5a4974cc3395f96b2a0fa9e6454202ef2c00d82b0e6c534b3b1d20f9a572

17 0x4991b763929b00241a1a9a68e00e90c5df087f90b3352c0f4d8094a51429524e

18 0x18b6b49c5650fb82e36e25fd4eb6decfdd40b46c37425e6597c7444a1b6afb4e

19 0x6868305b4f40654460aad63af3cb9151ab67c775eaac5e5df90d3aea58dee141

20 0x16bc90219a36063a22889db810730a8b719c267d538cd28fa7c0d04f124c8580

21 0x3628f9cf1fbe3eb559854e3b1c06a4cd6a26906b4e2d2e70616a493bba2dc574

22 0x64abcc6759f1ce1ab57d41e17c2633f717064e35a7233a6682f8cf8e9538afec

23 0x01
```

Appendix H. Point Compression

Point compression allows a shorter representation of affine points of an elliptic curve by exploiting algebraic relationships between the coordinate values based on the defining equation of the curve in question. Point decompression refers to the reverse process, where one tries and recover an affine point from its compressed representation and information on the domain parameters of the curve. Consequently, point compression followed by point decompression is the identity map.

The description below makes use of an auxiliary function (the parity function), which we first define for prime fields GF(p), with p odd, and then extend to all fields GF(q), where q is an odd prime power. We assume each finite field to be unambiguously defined and known from context.

Let y be a nonzero element of GF(q). If q:=p is an odd prime number, y and p-y can be uniquely represented as integers in the interval [1,p-1] and have odd sum p. Consequently, one can distinguish y from -y via the parity of this representation, i.e., via par(y):=y (mod 2). If q:=p^m, where p is an odd prime number and where m>0, both y and -y can be uniquely represented as vectors of length m, with coefficients in GF(p) (see Appendix B.2). In this case, the leftmost

nonzero coordinate values of y and -y are in the same position and have representations in [1,p-1] with different parity. As a result, one can distinguish y from -y via the parity of the representation of this coordinate value. This extends the definition of the parity function to any odd-size field GF(q), where one defines par(0):=0. The value of the parity function is commonly called the parity bit.

H.1. Point Compression for Weierstrass Curves

If P:=(X, Y) is an affine point of the Weierstrass curve W_{a,b} defined over the field GF(q), then so is -P:=(X, -Y). Since the defining equation Y^2=X^3+a*X+b has at most two solutions with fixed X-value, one can represent P by its X-coordinate and one bit of information that allows one to distinguish P from -P, i.e., one can represent P as the ordered pair compr(P):=(X, par(Y)). If P is a point of order two, one can uniquely represent P by its X-coordinate alone, since Y=0 and has fixed parity. Conversely, given the ordered pair (X, t), where X is an element of GF(q) and where t=0 or t=1, and the domain parameters of the curve W_{a,b}, one can use the defining equation of the curve to try and determine candidate values for the Y-coordinate given X, by solving the quadratic equation Y^2:=alpha, where alpha:=X^3+a*X+b. If alpha is not a square in GF(q), this equation does not have a solution in GF(q) and the ordered pair (X, t) does not correspond to a point of this curve. Otherwise, there are two solutions, viz. Y=sqrt(alpha) and -Y. If alpha is a nonzero element of GF(q), one can uniquely recover the Y-coordinate for which par(Y):=t and, thereby, the point P:=(X, Y). This is also the case if alpha=0 and t=0, in which case Y=0 and the point P has order two. However, if alpha=0 and t=1, the ordered pair (X, t) does not correspond to the outcome of the point compression function.

NOTE: the procedure above corrects an error in the point decompression procedure for Weierstrass curves defined over the prime field GF(p) of [SEC1], which erroneously converts a purported compressed point for which alpha=0 and t=1 (in the notation above), to the ordered pair (0,p).

We extend the definition of the point compression function to all points of the curve W_{a,b}, by associating the (non-affine) point at infinity O with any ordered pair compr(O):=(X,0), where X is any element of GF(q) for which alpha:=X^3+a*X+b is not a square in GF(q), and recover this point accordingly. In this case, the point at infinity O can be represented by any ordered pair (X,0) of elements of GF(q) for which X^3+a*X+b is not a square in GF(q). Note that this ordered pair does not satisfy the defining equation of the curve in question. An application may fix a specific suitable value of X or choose multiple such values and use this to encode additonal information. Further details are out of scope.

H.2. Point Compression for Montgomery Curves

If P:=(u, v) is an affine point of the Montgomery curve M_{A,B} defined over the field GF(q), then so is -P:=(u, -v). Since the defining equation B*v^2=u^3+A*u^2+u has at most two solutions with fixed u-value, one can represent P by its u-coordinate and one bit of information that allows one to distinguish P from -P, i.e., one can represent P as the ordered pair compr(P):=(u, par(v)). If P is a point of order two, one can uniquely represent P by its u-coordinate alone, since v=0 and has fixed parity. Conversely, given the ordered pair (u, t), where u is an element of GF(q) and where t=0 or t=1, and the domain parameters of the curve M_{A,B}, one can use the defining equation of the curve to try and determine candidate values for the v-coordinate given u, by solving the quadratic equation v^2:=alpha, where alpha:=(u^3+A*u^2+u)/B. If alpha is not a square in GF(q), this equation does not have a solution in GF(q) and the ordered pair (u, t) does not correspond to a point of this curve. Otherwise, there are two solutions, viz. v=sqrt(alpha) and -v. If alpha is a nonzero element of GF(q), one can uniquely recover the v-coordinate for which par(v):=t and, thereby, the affine point P:=(u, v). This is also the case if alpha=0 and t=0, in which case v=0 and the point P has order two. However, if alpha=0 and t=1, the ordered pair (u, t) does not correspond to the outcome of the point compression function.

We extend the definition of the point compression function to all points of the curve M_{A,B}, by associating the (non-affine) point at infinity O with the ordered pair compr(O):=(0,1) and recover this point accordingly. (Note that this corresponds to the case alpha=0 and t=1 above.) The point at infinity O can be represented by the ordered pair (0, 1) of elements of GF(q). Note that this ordered pair does not satisfy the defining equation of the curve in question.

H.3. Point Compression for Twisted Edwards Curves

If P:=(x, y) is an affine point of the twisted Edwards curve E_{a,d} defined over the field GF(q), then so is -P:=(-x, y). Since the defining equation a*x^2+y^2=1+d*x^2*y^2 has at most two solutions with fixed y-value, one can represent P by its y-coordinate and one bit of information that allows one to distinguish P from -P, i.e., one can represent P as the ordered pair compr(P):=(par(x), y). If P is a point of order one or two, one can uniquely represent P by its y-coordinate alone, since x=0 and has fixed parity. Conversely, given the ordered pair (t, y), where y is an element of GF(q) and where t=0 or t=1, and the domain parameters of the curve E_{a,d}, one can use the defining equation of the curve to try and determine candidate values for the x-coordinate given y, by solving the quadratic equation x^2:=alpha, where alpha:=(1-y^2)/(a-d*y^2).

(Here, observe that the denominator is nonzero for any point of E_{a,d}.) If alpha is not a square in GF(q), this equation does not have a solution in GF(q) and the ordered pair (t, y) does not correspond to a point of this curve. Otherwise, there are two solutions, viz. x=sqrt(alpha) and -x. If alpha is a nonzero element of GF(q), one can uniquely recover the x-coordinate for which par(x):=t and, thereby, the affine point P:=(x, y). This is also the case if alpha=0 and t=0, in which case x=0 and the point P has order one or two. However, if alpha=0 and t=1, the ordered pair (t, y) does not correspond to the outcome of the point compression function.

Note that the point compression function is defined for all points of the twisted Edwards curve E_{a,d}. Here, the identity element O:=(0,1) is associated with the compressed point compr(O):=(0,1). (Note that this corresponds to the case alpha=0 and t=0 above.)

We extend the definition of the compression function further, to also include a special marker element 'btm', by associating this marker element with the ordered pair compr(btm):=(1,1) and recover this marker element accordingly. (Note that this corresponds to the case alpha=0 and t=1 above.) The marker element 'btm' can be represented by the ordered pair (1,1) of elements of GF(q). Note that this ordered pair does not satisfy the defining equation of the curve in question.

Appendix I. Data Conversions

This section introduces various data conversion routines that are useful when representing integers, finite field elements, and curve points as binary or octet strings.

I.1. Strings and String Operations

The string over some alphabet S consisting of the symbols x_{l-1}, x_{l-2}, ..., x_1, x_0 (each in S), in this order, is denoted by str(x_{l-1}, x_{l-2}, ..., x_1, x_0) (or simply as x_{l-1} x_{l-2} ... x_1 x0, if the individual symbols can be uniquely identified). The length of this string (over S) is the number of symbols it contains (here: l). The empty string is the (unique) string of length l=0. Strings are commonly indicated by surrounding these by double quotation marks.

The right-concatenation of two strings X and Y (defined over the same alphabet) is the string Z consisting of the symbols of X (in the same order) followed by the symbols of Y (in the same order). The length of the resulting string Z is the sum of the lengths of X and Y. This string operation is denoted by Z:=X||Y. The string X is called a prefix of Z; the string Y a postfix of Z. The t-prefix of a string Z

of length l is its unique prefix X of length t; the t-postfix its unique postfix Y of length t (where in both cases t is an integer in the interval [0,l]). One can define these notions as well if t is outside the interval [0,l] by stipulating that a t-prefix or t-postfix is the empty string if t is negative and that it is the entire string Z if t is larger than l. Sometimes, a t-prefix of a string Z is denoted by trunc-left(Z,t); a t-postfix by trunc-right(Z,t). A string X is called a substring of Z if it is a prefix of some postfix of Z. The string resulting from prepending the string Y with X is the string X||Y. The symbols of a string Z of length l can be labelled from left to right, using consecutive integers in the interval [0,l) starting with zero, where each label identifies a symbol via its position in the string.

An octet (or byte) is an integer in the interval [0,256). An octet string is a string, where the alphabet is the set of all octets. A binary string (or bit string, for short) is a string, where the alphabet is the set {0,1} of binary digits. A hex digit is an integer in the interval [0,16), with the convention to denote the integers 10, 11, 12, 13, 14, and 15 by the symbols 'a', 'b', 'c', 'd', 'e', and 'f', respectively. A hexadecimal string (or hex string, for short) is a string, where the alphabet is the set of all hex digits. Note that the length of a string is defined in terms of the underlying alphabet, as are the operations in the previous paragraph.

Note that an octet z can be uniquely represented in base 16 as the integer z:=16*z1+z0, where z1 and z0 are hex digits, and, thereby, as the hexadecimal string z1||z0 of length two. This allows a concise description of octet strings as hex strings (commonly indicated by the "0x"-prefix).

An ASCII character is a symbol of the so-called ASCII alphabet: the set of symbols corresponding to the set of integers in the interval [0,128) according to the ASCII-table (see [RFC0020]). An ASCII string is a string, where the alphabet is the set of all ASCII characters. All ASCII characters corresponding to integers in the interval [33,126] are single printable characters (and can therefore be uniquely identified in a printable ASCII string). There is a 1-1 correspondence between ASCII characters and integers in the interval [0,128), thereby allowing each ASCII character to be uniquely represented by an octet.

I.2. Conversion between Bit Strings and Integers (BS2I, I2BS)

There is a 1-1 correspondence between bit strings of length l and integers in the interval [0, 2^l), where the bit string X:=str(x_{l-1}, x_{l-2}, ..., x_1, x_0) corresponds to the integer

x:=x_{l-1}*2^{l-1} + x_{l-2}*2^{l-2} + ... + x_1*2 + x_0*1. (If l=0, the empty bit string corresponds to the integer zero.) Note that while the mapping from bit strings to integers is uniquely defined, the inverse mapping from integers to bit strings is not, since any non-negative integer smaller than 2^t can be represented as a bit string of length at least t (due to leading zero coefficients in base 2 representation). The latter representation is called tight if the bit string representation has minimal length (the so-called bit-length of the integer in question). This defines the mapping BS2I from bit strings to integers and the mapping I2BS(x,l) from non-negative integers smaller than 2^l to bit strings of length l. Note that this also defines a 1-1 correspondence between bit strings of length four and hex digits, and the encoding of ASCII characters as bit strings of length eight (where the leftmost bit has the value zero), as stipulated in [RFC0020].

I.3. Conversion between Octet Strings and Integers (OS2I, I2OS)

There is a 1-1 correspondence between octet strings of length l and integers in the interval [0, 256^l), where the octet string X:=str(X_{l-1}, X_{l-2}, ..., X_1, X_0) corresponds to the integer x:=X_{l-1}*256^{l-1} + X^{l-2}*256^{l-2} + ... + X_1*256 + X_0*1. (If l=0, the empty string corresponds to the integer zero.) Note that while the mapping from octet strings to integers is uniquely defined, the inverse mapping from integers to octet strings is not, since any non-negative integer smaller than 256^t can be represented as an octet string of length at least t (due to leading zero coefficients in base 256 representation). The latter representation is called tight if the octet string representation has minimal length (the so-called byte-length of the integer in question). This defines the mapping OS2I from octet strings to integers and the mapping I2OS(x,l) from non-negative integers smaller than 256^l to octet strings of length l.

I.4. Conversion between Octet Strings and Bit Strings (OS2BS, BS2OS)

There is a 1-1 correspondence between octet strings of length l and bit strings of length 8*l, where the octet string X:=str(X_{l-1}, X_{l-2}, ..., X_1, X_0) corresponds to the right-concatenation of the 8-bit strings x_{l-1}, x_{l-2}, ..., x_1, x_0, where each octet X_i corresponds to the 8-bit string x_i according to the mapping of Appendix I.2 above. Note that the mapping from octet strings to bit strings is uniquely defined and so is the inverse mapping from bit strings to octet strings, if one prepends each bit string with the smallest number of 0 bits so as to result in a bit string of length divisible by eight (i.e., one uses pre-padding). This defines the mapping OS2BS from octet strings to bit strings and the corresponding mapping BS2OS from bit strings to octet strings. When we refer to a

specific bit position in an octet string, this indicates the corresponding position, when this octet string is viewed as a bit string using the OS2BS mapping above.

I.5. Conversion between Field Elements and Octet Strings (FE2OS, OS2FE)

There is a 1-1 correspondence between elements of the fixed finite field GF(q), where q:=p^m, where p is a prime number and where m>0, and vectors of length m, with coefficients in GF(p), where each element x of GF(q) is a vector (x_{m-1}, x_{m-2}, ..., x_1, x_0) according to the conventions of Appendix B.2. In this case, this field element can be uniquely represented by the right-concatenation of the octet strings X_{m-1}, X_{m-2}, ..., X_1, X_0, where each octet string X_i corresponds to the integer x_i in the interval [0,p-1] according to the mapping of Appendix I.3 above. Note that both the mapping from field elements to octet strings and the inverse mapping from octet strings to field elements are only uniquely defined if each octet string X_i has the same fixed size (e.g., the smallest integer l so that 256^l >= p) and if all integers are reduced modulo p. If so, the latter representation is called tight if l is minimal so that 256^l >= p. This defines the mapping FE2OS(x,l) from field elements to octet strings and the mapping OS2FE(X,l) from octet strings to field elements, where the underlying field is implicit and assumed to be known from context. In this case, the octet string has length l*m. (Observe that with tight representations, the parameter l is uniquely defined by the characteristic p of the field GF(q) in question.) The OS2FE(X,l) mapping is called strict if it operates as the OS2FE(X,l) function, except that it fails whenever it would require at least one modular reduction. Notice that the tight FE2OS mapping followed by the strict OS2FE mapping is the identity map (and, hence, OS2FE never fails in this case).

I.6. Conversion between Elements of Z_n and Octet Strings (ZnE2OS, OS2ZnE)

There is a 1-1 correspondence between elements of the set Z_n of integers modulo n and integers in the interval [0,n), where each element x of Z_n is uniquely represented by the integer x mod n. In this case, x mod n can be uniquely represented by the octet string X according to the mapping of Appendix I.3 above. Note that both the mapping from elements of Z_n to octet strings and the inverse mapping from octet strings to elements of Z_n are only uniquely defined if the octet string has a fixed size (e.g., the smallest integer l so that 256^l >= n) and if all integers are first reduced modulo n. If so, the latter representation is called tight if l is minimal so that 256^l >= n. This defines the mapping ZnE2OS(x,l) from elements of Z_n to octet strings and the mapping OS2ZnE(X,l) from octet strings

to elements of Z_n, where the underlying modulus n is implicit and assumed to be known from context. In this case, the octet string has length l. (Observe that with tight representations, the parameter l is uniquely defined by the parameter n in question.) The OS2ZnE(X,l) mapping is called strict if it operates as the OS2ZnE(X,l) function, except that it fails whenever it would require at least one modular reduction. Notice that the tight ZnE2OS mapping followed by the strict OS2ZnE mapping is the identity map (and, hence, ZnE2OS never fails in this case).

Note that if n is a prime number p, the conversions ZnE2OS and FE2OS are consistent, as are OS2ZnE and OS2FE. This is, however, no longer the case if n is a strict prime power.

The conversion rules for composite (i.e., non-prime) n values may be useful, e.g., when encoding RSA parameters (or elements of any other non-prime size set Z_n, for that matter).

I.7. Ordering Conventions

One can consider various representation functions, depending on bit-ordering and octet-ordering conventions.

The description below makes use of an auxiliary function (the reversion function), where the reverse of the string X:=str(x_{l-1}, x_{l-2}, ..., x_1, x_0) is defined to be the string X':=rev(X):=str(x_0, x_1, ..., x_{l-2}, x_{l-1}). Below, we use this reversion function with binary and octet strings.

We now describe representations in most-significant-bit first (msb) or least-significant-bit first (lsb) order and those in most-significant-byte first (MSB) or least-significant-byte first (LSB) order.

One distinguishes the following octet-string representations of integers and field elements:

1. MSB, msb: represent field elements and integers as above, yielding the octet string str(X_{l-1}, X_{l-2}, ..., X_1, X_0).

2. MSB, lsb: reverse the bit-order of each octet, viewed as 8-bit string, yielding the octet string str((rev(X_{l-1}), rev(X_{l-2}), ..., rev(X_1), rev(X_0)).

3. LSB, lsb: reverse the octet string and bit-order of each octet, yielding the octet string str(rev(X_{0}), rev(X_{1}), ..., rev(X_{l-2}), rev(X_{l-1})).

4. LSB, msb: reverse the octet string, yielding the octet string str(X_{0}, X_{1}, ..., X_{l-2}, X_{l-1}).

Thus, the 2-octet string "07e3" represents the integer 2019 (=0x07e3) in MSB/msb order, the integer 57,543 (0xe0c7) in MSB/lsb order, the integer 51,168 (0xc7e0) in LSB/lsb order, and the integer 58,119 (=0xe307) in LSB/msb order.

Note that, with the above data conversions, there is still some ambiguity as to how to represent an integer or a field element as a bit string or octet string (due to leading zeros). However, tight representations (as defined above) are non-ambiguous. (Note, in particular, that tightness implies that elements of GF(q) are always uniquely represented.)

I.8. Conversion Between Curve Points and Octet Strings

For each of the curve models we consider, each affine point is an ordered pair (X, Y) whose coordinates are elements of a finite field GF(q) and that satisfy the defining equation for the curve in question. Each compressed point is an ordered pair (X,t) (for Weierstrass curves and Montgomery curves) or (t, X) (for twisted Edwards curves) where X is an element of GF(q) and where t is an element of {0,1} (see Appendix H).

The affine point (X, Y) is represented by the ordered pair whose coordinates are the octet string representations of the elements X and Y of GF(q), respectively, using the tight FE2OS mapping of Appendix I.5. Note that, since we use a tight representation, this results in a pair of octet strings (each of length l*m), where the parameters l and m are uniquely defined by the field GF(q) in question. The inverse mapping results by converting the first and second coordinate of this pair (each an octet string of length l*m) to, respectively, the elements X and Y of GF(q) via the strict OS2FE mapping of Appendix I.5. Note that if it is not a priori known whether the input to this inverse mapping actually represents an affine curve point, one should check that the output (X,Y) -- if defined -- is indeed an affine point of the curve in question, where this operation fails if this is not the case. (This check involves simply checking whether the ordered pair (X,Y) satisfies the defining equation for this curve.)

The compressed point (X, t) or (t, X) is represented by the ordered pair whose coordinates are the octet string representations of the parity bit t in {0,1} and the element X of GF(q), respectively, using the tight FE2OS mapping of Appendix I.5. Note that, since we use tight representations, this results in an ordered pair of octet strings (of length 1 and l*m, respectively), where the parameters l

and m are uniquely defined by the field GF(q) in question. The inverse mapping results by converting the first and second coordinate of this pair (each an octet string, of length 1 and l*m, respectively) to, respectively, the element t of {0,1} and the element X of GF(q) via the strict OS2FE mapping of Appendix I.5, and representing this as the compressed point (X, t) or (t, X) according to the curve model in question. Note that if it is not a priori known whether the input to this inverse mapping actually represents a compressed curve point, one should check that the output (X, t) or (t, X) -- if defined -- is indeed a compressed point of the curve in question, using the point decompression process for this curve (see Appendix H), where this operation fails if this is not the case. (This check does include checking whether an element is a square in GF(q), but does not require actually computing square roots (see also the Note in Appendix K.1).)

NOTE 1: The representations of affine and compressed points above are as ordered pairs of octet strings. In practice, one often represents these as octet strings instead, via right-concatenation of its coordinates (in left-to-right order). Since each coordinate has known length, this operation is reversible. When appropriate, we refer to the latter as the octet (rather than the pair) representation of a point.

NOTE 2: The octet representation of compressed points above identifies the parity bit t of the curve point in question via the 1-octet representations of the integers 0 and 1. Obviously, other 1-1 mappings are also possible. As an example, with [SEC1], the parity bit t is represented by 0x02 or 0x03 depending on whether t=0 or t=1, respectively. The same [SEC1] specification represents affine points as above (as octet string), but prepends this with the 1-octet prefix 0x04, and represents the identity element of the curve as the 1-octet string 0x00. This variable-size point representation has the property that its 1-octet prefix identifies whether it encodes an affine curve point, a compressed point (including parity bit), or the identity element, while the remainder of this representation uniquely determines the curve point's value. While the description in [SEC1] only applies to Weierstrass curves, the description above applies to each of the curve models we consider (i.e., these apply to Montgomery curves and twisted Edwards curves as well) and also applies to curves defined over extension fields. Collectively, we simply refer to this as the "SEC1" point representation.

Note that elements of a prime field GF(p), where p is a 255-bit prime number, have a tight representation as a 32-octet string, where a fixed bit position is always set to zero. (This is the leftmost bit position of this octet string if one follows the MSB/msb

representation conventions.) This allows the parity bit of a compressed point (see Appendix H) to be encoded in this bit position and, thereby, allows a compressed point and an element of GF(p) to be represented by an octet string of the same length. This is called the "squeezed" point representation. (We will use this squeezed representation in Appendix J.) Obviously, other representations (e.g., those of elements of Z_n) may also have fixed bit values in certain positions, which can be used to squeeze-in additional information. Further details are out of scope.

Notice that elements of a prime field GF(p), where p is a prime number with bit-length m divisible by eight, have a tight representation as an (m/8)-octet string, but do not have a bit position that is always set to zero. Thus, in this case, one cannot represent a compressed point as an octet string of the same length as an element of GF(p). However, one can still encode this as an octet string of length (m/8)+1 (see Note 1 above). If one uses right-concatenation as in Note 1, but (for historial reasons) represents the parity bit t of the compressed point in question by 0x00 or 0x80 depending on whether t=0 or t=1, respectively, this is again called the "squeezed' representation (despite this being somewhat a misnomer, since each point is now represented as an octet string that is one octet longer than the tight representation of elements of GF(p)). Notice that this representation corresponds to the compressed point representation of Appendix I.8 as octet string, but with the bit-ordering in the 1-octet representation of t reversed. (Note that this puts the parity bit t in the leftmost bit position of the octet string if one follows the MSB/msb representation conventions.) We will use this squeezed represenation in Appendix O.

Appendix J. Representation Examples Curve25519 Family Members

We present some examples of computations using the curves introduced in Appendix E and Appendix G of this document. In each case, we indicate the values of P, k*P, and (k+1)*P, where P is a fixed multiple (here: 2019) of the base point of the curve in question and where the private key k is the integer

k 4546754475995463934419135116415656059529923676170206503367073967 7691372543056

(=0x6485b7e6 cd83e5c2 0d5dbfe4 f915494d 9cf5c65d 778c32c3 c08d5abd 15e29c50).

In the examples below, each curve point is represented using the "squeezed" point representation (see Appendix I.8), whereby each point is represented as a 32-octet string, where the ordering convention (see Appendix I.7) depends on the underlying curve model

in question. Here, points of a Weierstrass curve are represented in tight MSB/msb-order, points of a Montgomery curve in tight LSB/msb-order, and points of a twisted Edwards curve in tight LSB/lsb-order. For points that are a public key, the corresponding private keys are represented as 32-octet strings, using the same (tight) ordering conventions as with the public keys. For affine points, we also give the tight representation of each of its coordinates, using the same ordering conventions as used with the squeezed point representation. For further details, see the examples themselves.

J.1. Example with Curve25519

Pm=(u, v), k*Pm=(u1, v1), and (k+1)*Pm=(u2, v2) with Curve25519:

u 5302565753880801364561862039375446131953591537683081997498228933208825562375 0

(=0x753b7566 df35d574 4734142c 9abf931c ea290160 aa75853c 7f972467 b7f13246).

v 5332779809243646201304837030201994630082651145916190570914464552123369031308 6

(=0x75e676ce deee3b3c 12942357 22f1d884 ac06de07 330fb07b ae35ca26 df75417e).

u1 4203961881847433543933319291014302929445065173616660243524852844269171766805 6

(=0x5cf194be f0bdd6d6 be58e18a 8f16740a ec25f4b0 67f7980a 23bb6468 88bb9cd8).

v1 7698166198291735163093751722241272913088236885813432215648576219567913357634

(=0x110501f6 1dff511e d6c4e9b9 bfd5acbe 8bf043b8 c3e381dd f5771306 479ad142).

u2 3417511648237788235544013775257365183827376081862455752464312610182464621878

(=0x078e3e38 41c3e0d0 373e5454 ecffae33 2798b10a 55c72117 62629f97 f1394d36).

v2 4304698585363167161055383496878520419196717196793784253165625453996266399464 8

(=0x5f2bbb06 f7ec5953 2c2a1a62 21124585 1d2682e0 cc37307e
fbc17f7f 7fda8518).

As suggested in Appendix C.2, the v-coordinate of k*Pm can be indirectly computed from the u-coordinates of Pm, k*Pm, and (k+1)*Pm, and the v-coordinate of Pm, which allows computation of the entire point k*Pm (and not just its u-coordinate) if k*Pm is computed using the Montgomery ladder (as, e.g., [RFC7748] recommends), since that algorithm computes both u1 and u2 and the v-coordinate of the point Pm may be available from context.

The representation of k and the compressed representations of Pm and k*Pm in tight LSB/msb-order are given by

repr(k) 0x509ce215 bd5a8dc0 c3328c77 5dc6f59c 4d4915f9 e4bf5d0d c2e583cd e6b78564

repr(Pm) 0x4632f1b7 6724977f 3c8575aa 600129ea 1c93bf9a 2c143447 74d535df 66753b75;

repr(k*Pm) 0xd89cbb88 6864bb23 0a98f767 b0f425ec 0a74168f 8ae158be d6d6bdf0 be94f15c,

where the leftmost bit of the rightmost octet indicates the parity of the v-coordinate of the point of Curve25519 in question (which, in this case, are both zero, since v and v1 are even). See Appendix H.2 and Appendix I for further detail on (squeezed) point compression.

The scalar representation and (squeezed) point representation illustrated above are consistent with the representations specified in [RFC7748], except that in [RFC7748] only an affine point's u-coordinate is represented (i.e., the v-coordinate of any point is always implicitly assumed to have an even value) and that the representation of the point at infinity is not specified. Another difference is that [RFC7748] allows non-unique representations of some elements of GF(p), whereas our representation conventions do not (since tight).

A randomized representation (t1, t2) of the point k*Pm in tight LSB/msb order is given by

t1 40953131790112268570753571592444539842650348318985471658437762538294289253464

(=0x5844b232 8c4586dc 62f593c5 599c2a8c e61ba893 bb052de6 77510a42 b3a68a5a)

t2 45185609833288940742127800462815081444925990202338853392908884892762543098088 1

(=0x11598452 e65138dc ce948d7e d8f46a18 b640722c 8e170957 751b7729 1b26e663),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.2 with the default square root function.

This representation can also be expressed in tight LSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(0,0) and where

u1 54518733982984694553806836404858182101845571463259598899020004161172542370 99

(=0xabab17e4 f1dbafb1 ede0c4b3 bedb7734 9c85f2a7 917c5edf ad4bd96a a7a60d0c)

u2 23626346884803127022385404664577298006457681657894934495776188172480447798 47

(=0x47099c3e 9b5cc8fe eaac5db0 6fb413fa b3ef4516 7bfcdc4b 8368f22e 2f343905),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.2 (with the default square root function).

J.2. Example with Edwards25519

Pe=(x, y), k*Pe=(x1, y1), and (k+1)*Pe=(x2, y2) with Edwards25519:

x 2530166234870213609260226823618336108586393247559312047538295905336538722325 2

(=0x37f03bc0 1070ed12 d3218f8b ba1abb74 fd6b94eb 62033d09 83851e21 d6a460d4).

y 5443474914517576279855043665674856841109970216812159209060850157894201947336 0

(=0x7858f9e7 6774ed8e 23d614d2 36715fc7 56813b02 9aa13c18 960705c5 b3a30fd0).

x1 4296696779658546073386172486569954827997873046076602508744450281241655728487 3

(=0x5efe7124 465b5bdb b364bb3e e4f106e2 18d59b36 48f4fe83
c11afc91 785d7e09).

y1 46006463385134057167371782068441558951541960707376246310705917936
352255317084

(=0x65b6bc49 985badaf bc5fdd96 fb189502 35d5effd 540b439d
60508827 80bc945c).

x2 42629294840915692510487991904657367226900127896202625319538173473
104931719808

(=0x5e3f536a 3be2364a 1fa775a3 5f8f65ae 93f4a89d 81a04a2e
87783748 00120a80).

y2 29739282897206659585364020239089516293417836047563355347155817358
737209129078

(=0x41bfd66e 64bdd801 c581a720 f48172a8 187445fa 350924a2
c92c791e 38d57876).

The representation of k and the compressed representations of Pe and k*Pe in tight LSB/lsb-order are given by

repr(k) =0x0a3947a8 bd5ab103 c34c31ee ba63af39 b292a89f 27fdbab0
43a7c1b3 67eda126;

repr(Pe) =0x0bf0c5cd a3a0e069 183c8559 40dc816a e3fa8e6c 4b286bc4
71b72ee6 e79f1a1e;

repr(k*Pe) =0x3a293d01 e4110a06 b9c2d02a bff7abac 40a918df 69bbfa3d
f5b5da19 923d6da7,

where the rightmost bit of the rightmost octet indicates the parity of the x-coordinate of the point of Edwards25519 in question (which, in this case, are zero and one, respectively, since x is even and x1 is odd). See Appendix H.3 and Appendix I for further detail on (squeezed) point compression.

The scalar representation and (squeezed) point representation illustrated above are fully consistent with the representations specified in [RFC8032]. Note that, contrary to [RFC7748], [RFC8032] requires unique representations of all elements of GF(p).

A randomized representation (t1, t2) of the point k*Pe in tight LSB/lsb order is given by

t1 5779130170831636419496342190171901821702887766487253959359775042751939925404 0

(=0x181a32c5 10e06dbc ea321882 f3519055 535e289e 8faac654 82e26f61 aded23fe)

t2 4548814079409197184266085731253774016862550682106242458840547971622048028797 4

(=0x672e36c5 ae353073 cdfac343 e8297b05 1b010d0f 5b1016db dd4baf54 28068926),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.3 with the default square root function and underlying isomorphic mapping between Edwards25519 and Curve25519 of Appendix E.2.

This representation can also be expressed in tight LSB/lsb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(0,1) and where

u1 2244626522139140131658613866265237242854180727747413335904619130523458519264 4

(=0x2311ee45 c788a81b 7fcd7ae1 c6982d7b 537011fd d49e2eb4 62b9c08c 5344058c)

u2 1039512154902269015527669019928086231946046501815308223629026508474142603215

(=0xf3ed475b fd95335c 3a0ceb7e 319f8d3c cc651d5b 17eb4439 e3b25693 0bea3240),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.3 (with the default square root function).

J.3. Example with Wei25519

Pw=(X, Y), k*Pw=(X1, Y1), and (k+1)*Pw=(X2, Y2) with Wei25519:

X 1442829445970261517109495872419182536844592048828396529516309466278387923933 8

(=0x1fe62011 89e0801e f1debed7 456a3dc7 94d3ac0b 55202fe7 2a41cf12 629e56aa).

Y 5332779809243646201304837030201994630082651145916190570914464552123369031308 6

(=0x75e676ce deee3b3c 12942357 22f1d884 ac06de07 330fb07b ae35ca26 df75417e).

X1 3442255739368936964809531240580393343360656847619747755429333773387341283644

(=0x079c3f69 9b688181 69038c35 39c11eb5 96d09f5b 12a242b4 ce660f13 3368c13c).

Y1 7698166198291735163093751722241272913088236885813432215648576219567913357634

(=0x110501f6 1dff511e d6c4e9b9 bfd5acbe 8bf043b8 c3e381dd f5771306 479ad142).

X2 2271619318779048747280584461003868315937237352613588309237390994483465305741 5

(=0x3238e8e2 ec6e8b7a e1e8feff 97aa58dd d2435bb5 0071cbc2 0d0d4a42 9be67187).

Y2 4304698585363167161055383496878520419196717196793784253165625453996266399464 8

(=0x5f2bbb06 f7ec5953 2c2a1a62 21124585 1d2682e0 cc37307e fbc17f7f 7fda8518).

The representation of k and the compressed representations of Pw and k*Pw in tight MSB/msb-order are given by

repr(k) =0x6485b7e6 cd83e5c2 0d5dbfe4 f915494d 9cf5c65d 778c32c3 c08d5abd 15e29c50;

repr(Pw) =0x1fe62011 89e0801e f1debed7 456a3dc7 94d3ac0b 55202fe7 2a41cf12 629e56aa;

repr(k*Pw) =0x079c3f69 9b688181 69038c35 39c11eb5 96d09f5b 12a242b4 ce660f13 3368c13c,

where the leftmost bit of the leftmost octet indicates the parity of the Y-coordinate of the point of Wei25519 in question (which, in this case, are both zero, since Y and Y1 are even). See Appendix H.1 and Appendix I for further detail on (squeezed) point compression.

The scalar representation is consistent with the representations specified in [SEC1]; the (squeezed) point representation illustrated above is "new". For completeness, we include a SEC1-consistent

representation of the point Pw in affine format and in compressed format below.

The SEC1-compliant affine representation of the point Pw in tight MSB/msb-order is given by

aff(Pw) =0x04 1fe62011 89e0801e f1debed7 456a3dc7 94d3ac0b 55202fe7 2a41cf12 629e56aa

75e676ce deee3b3c 12942357 22f1d884 ac06de07 330fb07b ae35ca26 df75417e,

whereas the SEC1-compliant compressed representation of the point Pw in tight MSB/msb-order is given by

compr(Pw) =0x02 1fe62011 89e0801e f1debed7 456a3dc7 94d3ac0b 55202fe7 2a41cf12 629e56aa;

The SEC1-compliant uncompressed format aff(Pw) of an affine point Pw corresponds to the right-concatenation of its X- and Y-coordinates, each in tight MSB/msb-order, prepended by the string 0x04, where the reverse procedure is uniquely defined, since elements of GF(p) have a unique fixed-size representation. The (squeezed) compressed format repr(Pw) corresponds to the SEC1-compliant compressed format by extracting the parity bit t from the leftmost bit of the leftmost octet of repr(Pw), replacing the bit position by the value zero, and prepending the octet string with 0x02 or 0x03, depending on whether t=0 or t=1, respectively, where the reverse procedure is uniquely defined, since GF(p) is a 255-bit prime field. For further details, see [SEC1]. Note that, due to the bit-length of the prime p, the squeezed compressed format repr(Pw) is one octet shorter than the SEC1-compliant compressed format compr(Pw).

A randomized representation (t1, t2) of the point k*Pw in tight MSB/msb order is given by

t1 4463634459888897340934462804841221072830592062433079553888422315222879589959 0

(=0x62af4697 4dd469ac 96c64809 c16c8517 b6a0cee5 40ba0e2e 6dd2b36a fcc75ec6)

t2 2138901666102286131057927107083859617122112817447562160611193088805960310756 1

(=0x2f49c121 8fed7912 031157ee ae066507 a972320b 6180e267 4025b006 2e67bee9),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.1 with the default square root function.

This representation can also be expressed in tight MSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(1,0) and where

u1 52009283397096628981011768915795130293644642426523008816265117106436465991934

(=0x72fc3612 b18d2644 c2a85b3b dd66cd58 07ebf07b 2131b77d 6d7579da 5efba0fe)

u2 13400594985642565311540583887811555126397683953565069725078991786686428785368

(=0x1da077cd 6fa87515 731029a8 bd88da6a 34e38b83 51191edf 8a3b92d7 ba24aad8),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.1 (with the default square root function).

J.4. Example with Wei25519.2

Pw2=(X, Y), k*Pw2=(X1, Y1), and (k+1)*Pw2=(X2, Y2) with Wei25519.2:

X 1783049320995114833100801470107998886263453139413723543857183638922719845976 3

(=0x276bb396 d766b695 bfe60ab1 3c0260dd c09f5bcf 7b3ca47c f21c8672 d1ecaf73).

Y 2106449201293389610533824194047777846186606048140822212297983620613707578964 0

(=0x2e921479 5ad47af7 784831de 572ed8e9 7e20e137 cc67378c 184ca19f f9136f48).

X1 6547098895168646197978963236237775946468834215401735383493920379139281908968

(=0x0e7986d2 e94354ab 8abd8806 3154536a 4dcf8e6e 65557183 e242192d 3b87f4e8).

Y1 5148959049429218356253579057948003322904327153929727588881712522735262330110

(=0x0b623521 c1ff84bc 1522ff26 3376796d be77fcad 1fcabc28
98f1be85 d7576cfe).

X2 837417885015172009428261536776821209988540865517516630613749353883494226693

(=0x01d9f633 b2ac2606 9e6e93f7 6917446c 2b27c16f 729121d7
709c0a58 00ef9b05).

Y2 42567334190622848157611574766896093933050043101247319937794684825168161540336

(=0x5e1c41e1 fb74e41b 3a19ce50 e1b2caf7 7cabcbb3 0c1c1474
a4fd13e6 6c4c08f0).

The representation of k and the compressed representations of Pw2 and k*Pw2 in tight MSB/msb-order are given by

repr(k) =0x6485b7e6 cd83e5c2 0d5dbfe4 f915494d 9cf5c65d 778c32c3
c08d5abd 15e29c50;

repr(Pw2) =0x276bb396 d766b695 bfe60ab1 3c0260dd c09f5bcf 7b3ca47c
f21c8672 d1ecaf73;

repr(k*Pw2) =0x0e7986d2 e94354ab 8abd8806 3154536a 4dcf8e6e 65557183
e242192d 3b87f4e8,

where the leftmost bit of the leftmost octet indicates the parity of the Y-coordinate of the point of Wei25519.2 in question (which, in this case, are both zero, since Y and Y1 are even). See Appendix Appendix H.1 and Appendix I for further detail on (squeezed) point compression.

A randomized representation (t1, t2) of the point k*Pw2 in tight MSB/msb order is given by

t1 41666967235492814867975859880366011240543115979327816187936189858804289581274

(=0x5c1eaaef 80f9d4af 33c119fc c99acd58 f81e7d69 999c7048
e4043a77 87a930da)

t2 36111527116239160808309656017933739105961565127912319992118531180247832114098

(=0x4fd66668 e7174775 de44c852 92df8cfe b9832ef8 2570b3b8
fe5ec21a b2d4b3b2),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.1 with the default square root function.

This representation can also be expressed in tight MSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(1,0) and where

u1 13821549931386245347291517474076545480085862756377272673862176256261157017834

(=0x1e8eb854 2ce139f7 fdbf2059 ac257c89 d7e2e5fe 9c4b97e6 7656d42c 590bd8ea)

u2 52875019268539868510428902125104979140510466568127530408077706116783659458600

(=0x0bb09eba b0470a84 0ce1ba90 0aeab208 7e8d4760 1309d7af e3712e1f 2232a028),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.1 (with the default square root function).

J.5. Example with Wei25519.-3

Pw3=(X, Y), k*Pw3=(X1, Y1), and (k+1)*Pw3=(X2, Y2) with Wei25519.-3:

X 14780197759513083469009623947734627174363231692126610860256057394455099634096

(=0x20ad4ba4 612f0586 221787b0 d01ba46c d1d8cd5a 0348ef00 eb4c9272 03ca71b0).

Y 45596733430378470319805536538617129933663237960146030424392249401952949482817

(=0x64ced628 e982648e 4bfcf30c 71c4d267 ba48b0ce fee20062 b43ef4c9 73f7b541).

X1 47362979975244556396292400751828272600887612546997532158738958926 60745725532

(=0x0a78a650 a39995ef dcf4de88 940d4ce9 5b2ca35c c5d70e06 63b8455e 2e04e65c).

Y1 30318112837157047703426636957515037640997356617656007157255559136153389790354

(=0x4307719a 20d08741 58d5889e 8c8ec27e 246b0342 55f8fd62
dbc9ca09 e79c7492).

X2 237789420858737864335060630220598532128802964996223282012954465802935916643363

(=0x3492677e 6ae9d1c3 e08f908b 61033f3d 4e8322c9 fba6da81
2c95b067 9b1486eb).

Y2 448463663946517362483167491706870532726828478230182874390565379919695111504 94

(=0x632624d4 ab94c83a 796511c0 5f5412a3 876e56d2 ed18eca3
21b95bef 7bf9939e).

The representation of k and the compressed representations of Pw3 and k*Pw3 in tight MSB/msb-order are given by

repr(k) =0x6485b7e6 cd83e5c2 0d5dbfe4 f915494d 9cf5c65d 778c32c3
c08d5abd 15e29c50;

repr(Pw3) =0xa0ad4ba4 612f0586 221787b0 d01ba46c d1d8cd5a 0348ef00
eb4c9272 03ca71b0;

repr(k*Pw3) =0x0a78a650 a39995ef dcf4de88 940d4ce9 5b2ca35c c5d70e06
63b8455e 2e04e65c,

where the leftmost bit of the leftmost octet indicates the parity of the Y-coordinate of the point of Wei25519.-3 in question (which, in this case, are one and zero, respectively, since Y is odd and Y1 is even). See Appendix H.1 and Appendix I for further detail on (squeezed) point compression.

A randomized representation (t1, t2) of the point k*Pw3 in tight MSB/msb order is given by

t1 57371493761359660152568068464215566709721747496481624688988981222729740900 8259

(=0x7ed71d5f 566d2259 99bdb404 bfb9d6cf d2e86ccb 1894d4a6
c75e3c69 e5eb0283)

t2 26994578132458018981514201566389293572241945386392728723557891665397640090729

(=0x3bae63c8 70f60de0 c2e35f94 d24220f1 bb6efd00 37625869
f84923de ff4c5469),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.1 with the default square root function.

This representation can also be expressed in tight MSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(1,1) and where

u1 27359251097960067402783747714635503703273219507815338913481162438438522584713

(=0x3c7cc990 81eed784 9ca746d7 c479a902 ce9de65f 1150e7b9 c87d08d2 9785fe89)

u2 27148876502474775570472910326017705974534917128214682345800069381584030663589

(=0x3c05b835 1283fca7 705eba74 1e6b853e db3ed5dc d1891daa c1643d8d d63a03a5),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.1 (with the default square root function).

Appendix K. Auxiliary Functions

This section illustrates how one could implement common routines, such as taking square roots and inverses in finite fields, and how to map field elements to curve points and to curve points that avoid outlier points in the small subgroup.

K.1. Square Roots in GF(q)

Square roots are easy to compute in GF(q) if q = 3 (mod 4) (see Appendix K.1.1) or if q = 5 (mod 8) (see Appendix K.1.2). Details on how to compute square roots for other values of q are out of scope. If square roots are easy to compute in GF(q), then so are these in GF(q^2).

NOTE: If one wishes to check whether an element is a square in GF(q), rather than actually compute square roots, more efficient methods can be used. As an example, if GF(q) is a prime field (i.e., q:=p), one can efficiently check whether an element y is a square in GF(p) by computing its Legendre symbol (y/p) (see Section 2.4.5 of [Handbook]). Details on how to efficiently check whether an element is a square in GF(q) for other values of q are out of scope. If checking whether an element is a square is easy in GF(q), then so it is in GF(q^2).

K.1.1. Square Roots in GF(q), where q = 3 (mod 4)

If y is a nonzero element of GF(q) and z:=y^{(q-3)/4}, then y is a square in GF(q) only if y*z^2=1.

a. If y*z^2=+1, z is a square root of 1/y and y*z is a square root of y in GF(q);

b. If y*z^2=-1, z is a square root of -1/y and y*z is a square root of -y in GF(q).

(Note that the field element -1 is a non-square in GF(q).)

K.1.2. Square Roots in GF(q), where q = 5 (mod 8)

If y is a nonzero element of GF(q) and z:=y^{(q-5)/8}, then y is a square in GF(q) only if y^2*z^4=1.

a. If y*z^2=+1, z is a square root of 1/y and y*z is a square root of y in GF(q);

b. If y*z^2=-1, i*z is a square root of 1/y and i*y*z is a square root of y in GF(q);

c. If y*z^2=+i, z is a square root of i/y and y*z is a square root of i*y in GF(q);

d. If y*z^2=-i, z is a square root of -i/y and i*y*z is a square root of i*y in GF(q).

Here, i is an element of GF(q) for which i^2=-1 (e.g., i:=2^{(q-1)/4}). This field element is a non-square in GF(q) and can be precomputed.

K.2. Inversion

If y is an integer and gcd(y,n)=1, one can efficiently compute 1/y (mod n) via the extended Euclidean Algorithm (see Section 2.2.5 of [GECC]). One can use this algorithm as well to compute the inverse of a nonzero element y of a prime field GF(p), since gcd(y,p)=1.

The inverse of a nonzero element y of GF(q) can be computed as

1/y:=y^{q-2} (since y^{q-1}=1 by Fermat's Little Theorem).

If inverses are easy to compute in GF(q), then so are these in GF(q^2). Further details are out of scope.

The inverses of two nonzero elements y1 and y2 of GF(q) can be computed by first computing the inverse z of y1*y2 and by subsequently computing y2*z=:1/y1 and y1*z=:1/y2.

NOTE 1: This method can be used to compute the inverse of a nonzero element y of GF(q) indirectly, as lambda*(lambda*y)^{-1}, where lambda is a random nonzero element of GF(q). This yields an inversion routine (commonly called "blinded inversion") where the inversion operation itself does not leak information on y.

NOTE 2: This method can also be used to compute the inverse and a square root, respectively, of two nonzero elements x and y of GF(q) (where y is a square in GF(q)) by first computing a square root z of 1/(y*x^2) (see Appendix K.1) and by subsequently computing a square root of y as x*y*z and the inverse of x as x*y*z^2.

K.3. Mappings to Curve Points

One can map elements of GF(q) that are not a square in GF(q) to points of a Weierstrass curve (see Appendix K.3.1), to points of a Montgomery curve (see Appendix K.3.2), or to points of a twisted Edwards curve (see Appendix K.3.3), under some mild conditions on the domain parameters. Full details on mappings that apply if these conditions are not satisfied are out of scope.

K.3.1. Mapping to Points of Weierstrass Curve

The description below assumes that the domain parameters a and b of the Weierstrass curve W_{a,b} are nonzero. For ease of exposition, we define f(z):=z^3+a*z+b. (Note that for an affine point (X,Y) of W_{a,b} one has Y^2=f(X).)

If t is an element of GF(q) that is not a square in GF(q) and that is unequal to -1, then the element X:=(-b/a)*(1+1/(t+t^2)) is the unique solution of the equation f(t*X)=t^3*f(X) and is nonzero. Consequently, either X or X':=t*X is the x-coordinate of an affine point of W_{a,b}, depending on whether f(X) is a square in GF(q).

a. If f(X) is a square in GF(q) and Y:=sqrt(f(X)), then t is mapped to the point P(t):=(X, Y);

b. If f(X) is not a square in GF(q) and Y':=sqrt(f(X')), then t is mapped to the point P(t):=(X', -Y').

Formally, this mapping is not properly defined, since a nonzero square y:=x^2 in GF(q) has two solutions, viz. x and -x; it is properly defined, however, if one designates for each element of GF(q) that is a square in GF(q) precisely one square root as "the"

square root of this element. Note that always picking the square root with zero parity (see Appendix H) satisfies this condition (henceforth called the default square root function).

If -1 is not a square in GF(q), this element is mapped to the point at infinity O of W_{a,b}.

The set of points of W_{a,b} that arises this way has size roughly 3/8 of the order of the curve and each such point arises as image of one or two t values. Further details are out of scope.

NOTE 1: If -1 is not a square in GF(q), the mapping above yields the point at infinity for t=-1. One can modify this mapping, by mapping the element -1 to any suitable point P0 of W_{a,b} (e.g., its base point G or any other affine point) and leaving the remainder of the mapping the same. Suitability of such a modification is application-specific. Details are out of scope.

NOTE 2: The description above assumes that the domain parameters a and b of the Weierstrass curve W_{a,b} are nonzero. If this is not the case, one can often find an isogenous curve W_{a',b'} for which the domain parameters a' and b' are nonzero. If so, one can map elements of GF(q) that are not a square in GF(q) to points of W_{a,b} via function composition, where one uses the mapping above to arrive at a point of W_{a',b'} and where one subsequently uses the dual isogeny from W_{a',b'} to W_{a,b} to arrive at a point of W_{a,b}. As an example, one can show that if a is zero and if -4*b is a cube in GF(q) (such as is the case with, e.g., the "BitCoin" curve secp256k1 [SEC2]), this curve is 3-isogenous to a curve with this property and the strategy above applies (for an example with secp256k1, see Appendix L). Further details are out of scope.

K.3.2. Mapping to Points of Montgomery Curve

The description below assumes that the domain parameter A of the Montgomery curve M_{A,B} is nonzero. For ease of exposition, we define f(z):=z^3+A*z^2+z. (Note that for an affine point (u,v) of M_{A,B} one has B*v^2=f(u).)

If t is an element of GF(q) that is not a square in GF(q) and that is unequal to -1, then the element u:=-(1+1/t)/A is the unique nonzero solution of the equation f(t*u)=t^3*f(u). Consequently, either u or u':=t*u is the u-coordinate of an affine point of M_{A,B}, depending on whether f(u)/B is a square in GF(q).

a. If f(u)/B is a square in GF(q) and v:=sqrt(f(u)/B), then t is mapped to the point P(t):=(u, v);

b. If f(u)/B is a not a square in GF(q) and v':=sqrt(f(u')/B), then t is mapped to the point P(t):=(u', -v').

As before, formally, this mapping is not properly defined, since a nonzero square y:=x^2 in GF(q) has two solutions, viz. x and -x; it is properly defined, however, if one designates for each element of GF(q) that is a square in GF(q) precisely one square root as "the" square root of this element. Note that always picking the square root with zero parity (see Appendix H) satisfies this condition (henceforth called the default square root function).

If -1 is not a square in GF(q), this element is mapped to the point at infinity O of M_{A,B}.

The set of points of M_{A,B} that arises this way has size roughly 1/2 of the order of the curve and each such point arises as image of precisely one t value. Further details are out of scope.

NOTE 1: If -1 is not a square in GF(q), the mapping above yields the point at infinity for t=-1. One can modify this mapping, by mapping the element -1 to any suitable point P0 of M_{a,b} (e.g., its base point G or any other affine point) and leaving the remainder of the mapping the same. Suitability of such a modification is application-specific. Details are out of scope.

NOTE 2: The description above assumes that the domain parameter A of the Montgomery curve M_{A,B} is nonzero. If this is not the case, the curve is a Weierstrass curve for which the domain parameter b is zero and Note 2 of Appendix K.3.1 applies. If q = 3 (mod 4), an even simpler approach is possible, where one modifies the construction above and simply takes u:=t and u':=-t (which works, since -1 is not a square in GF(q) and f(-t)=-f(t)). In this case, this construction can be extended to all elements t of GF(q) and, if so, yields a 1-1 mapping between GF(q) and all affine curve points.

K.3.3. Mapping to Points of Twisted Edwards Curve

One can map elements of GF(q) that are not a square in GF(q) to points of the twisted Edwards curve E_{a,d} via function composition, where one uses the mapping of Appendix K.3.1 to arrive at a point of the Weierstrass curve W_{a,b} and where one subsequently uses the isomorphic mapping between twisted Edwards curves and Weierstrass curves of Appendix D.3 to arrive at a point of E_{a,d}. Another mapping is obtained by function composition, where one instead uses the mapping of Appendix K.3.2 to arrive at a point of the Montgomery curve M_{A,B} and where one subsequently uses the isomorphic mapping between twisted Edwards curves and Montgomery curves of Appendix D.1 to arrive at a point of E_{a,d}. Obviously, one can use function

composition (now using the respective pre-images - if these exist) to realize the pre-images of either mapping.

K.4. Mappings to High-Order Curve Points

Appendix K.3 described how one can map elements of GF(q) that are not a square in GF(q) to points of a Weierstrass curve, to points of a Montgomery curve, or to points of a twisted Edwards curve, under some mild conditions on the domain parameters. Below, we use the mappings of that appendix and the parity function par(.) specified in Appendix H to construct mappings to high-order curve points only (i.e., mappings that avoid points in the small subgroup, see Appendix B.1). We consider mappings to high-order points of a Weierstrass curve (see Appendix K.4.1), to high-order points of a Montgomery curve (see Appendix K.4.2), and to high-order points of a twisted Edwards curve (see Appendix K.4.3). As before, full details on mappings that apply if the mild conditions on the domain parameters are not satisfied are out of scope.

K.4.1. Mapping to High-Order Points of Weierstrass Curve

The description below assumes that the domain parameters a and b of the Weierstrass curve W_{a,b} are nonzero. For ease of exposition, we define f(z):=z^3+a*z+b. (Note that for an affine point (X,Y) of W_{a,b} one has Y^2=f(X).)

If t is an element of GF(q) that is not a square in GF(q) and that is unequal to -1, the mapping of Appendix K.3.1 yields an affine point P(t):=(X, Y) of W_{a,b}. Let P0:=(X0, Y0) be a fixed affine point of W_{a,b} for which neither P0, P0 + P(t), nor P0 - P(t) is in the small subgroup of W_{a,b} (for any non-square element t<>-1 of GF(q)). (Note that this implies that P0 and P(t) are distinct affine points of the curve and that these are not each other's inverse.) For binary digit s, the point Q(t,s) is now defined as follows:

a. If par(Y0*Y)=s, then the pair (t,s) is mapped to the point Q(t,s):=P0 + P(t);

b. If par(Y0*Y)<>s, then the pair (t,s) is mapped to the point Q(t,s):=P0 - P(t).

Note that this mapping is properly defined as long as the fixed point P0 (the so-called "curve offset") alluded to above indeed exists. In cases of practical interest that we are aware of, this is indeed the case (see, e.g., Table 1).

If -1 is not a square in GF(q), the pair (-1,s) is mapped to the affine point P0 of W_{a,b} (irrespective of the value of s).

The set of points of W_{a,b} that arises this way has size roughly 3/8 of the order of the curve and each such point arises as image of up to four values of the pair (t,s). Further details are out of scope.

From the group law for Weierstrass curves (see Appendix C.1) it follows that one can express the coordinates of Q(t,s), with t<>-1, in terms of the X-coordinates of P0 and P(t) and the product of their Y-coordinates. (Here, observe that Y0*Y is a square root of f(X0)*f(X).) Thus, Q(t,s) can be computed without the need to fully compute P(t).

K.4.2. Mapping to High-Order Points of Montgomery Curve

The description below assumes that the domain parameters A and B of the Montgomery curve M_{A,B} are nonzero. For ease of exposition, we define f(z):=z^3+A*z^2+z. (Note that for an affine point (u,v) of M_{A,B} one has B*v^2=f(u).)

If t is an element of GF(q) that is not a square in GF(q) and that is unequal to -1, the mapping of Appendix K.3.2 yields an affine point P(t):=(u, v) of M_{A,B}. Let P0:=(u0, v0) be a fixed affine point of M_{A,B} for which neither P0, P0 + P(t), nor P0 - P(t) is in the small subgroup of M_{A,B} (for any non-square element t<>-1 of GF(q)). (Note that this implies that P0 and P(t) are distinct affine points of the curve and that these are not each other's inverse.) For binary digit s, the point Q(t,s) is now defined as follows:

a. If par(B*v0*v)=s, then the pair (t,s) is mapped to the point Q(t,s):=P0 + P(t);

b. If par(B*v0*v)<>s, then the pair (t,s) is mapped to the point Q(t,s):=P0 - P(t).

Note that this mapping is properly defined as long as the fixed point P0 (the so-called "curve offset") alluded to above indeed exists. In cases of practical interest that we are aware of, this is indeed the case (see, e.g., Table 1).

If -1 is not a square in GF(q), the pair (-1,s) is mapped to the affine point P0 of M_{A,B} (irrespective of the value of s).

The set of points of M_{A,B} that arises this way has size roughly 1/2 of the order of the curve and each such point arises as image of up to two values of the pair (t,s). Further details are out of scope.

From the group law for Montgomery curves (see Appendix C.2) it follows that one can express the coordinates of Q(t,s), with t<>-1, in terms of the u-coordinates of P0 and P(t) and the product of their v-coordinates. (Here, observe that B*v0*v is a square root of f(u0)*f(u).) Thus, Q(t,s) can be computed without the need to fully compute P(t).

Curve	Fixed curve offset P0	Non-Square
NIST P-224 [FIPS-186-4]	Base point (Gx,Gy)	11
NIST P-256 [FIPS-186-4]	P0:=(0,y), y even	-1
NIST P-384 [FIPS-186-4]	P0:=(0,y), y even	-1
NIST P-521 [FIPS-186-4]	P0:=(0,y), y even	-1
brainpoolP224r1 [RFC5639]	Base point (Gx, Gy)	-1
brainpoolP256r1 [RFC5639]	Base point (Gx, Gy)	-1
brainpoolP320r1 [RFC5639]	Base point (Gx, Gy)	-1
brainpoolP384r1 [RFC5639]	Base point (Gx, Gy)	-1
brainpoolP512r1 [RFC5639]	P0:=(3,y), y even	-1
Curve25519 [RFC7748]	P0:=(90,v), v even	2
Wei25519 [Appendix E.3]	P0:=(3,y), y even	2
Wei25519.2 [Appendix G.3]	P0:=(244,y), y even	2
Wei25519.-3 [Appendix G.3]	P0:=(41,y), y even	2
Curve448 [RFC7748]	P0:=(50,v), v even	-1
Wei448 [Appendix M.3]	P0:=(18,y), y even	-1
Wei448.1 [Appendix N.3]	P0:=(10,y), y even	-1
Wei448.-3 [Appendix N.3]	P0:=(8,y), y even	-1
secp256k1.m [Appendix L.3]	P0:=(0,y), y even	-1

Table 1: Fixed curve offsets for mappings that avoid low-order points, for some curves of practical interest, including listing of fixed non-square elements of their underlying finite fields.

K.4.3. Mapping to High-Order Points of Twisted Edwards Curve

One can map elements of GF(q) that are not a square in GF(q) to points of the twisted Edwards curve E_{a,d} via function composition, where one uses the mapping of Appendix K.4.1 to arrive at a point of the Weierstrass curve W_{a,b} that is not in the small subgroup and where one subsequently uses the isomorphic mapping between twisted Edwards curves and Weierstrass curves of Appendix D.3 to arrive at a point of E_{a,d} with this property. Another mapping is obtained by function composition, where one instead uses the mapping of Appendix K.4.2 to arrive at a point of the Montgomery curve M_{A,B} that does not have low order and where one subsequently uses the isomorphic mapping between twisted Edwards curves and Montgomery curves of Appendix D.1 to arrive at a point of E_{a,d} with this

property. Obviously, one can use function composition (now using the respective pre-images - if these exist) to realize the pre-images of either mapping.

K.5. Randomized Representation of Curve Points

The mappings of Appendix K.3 allow one to represent a curve point Q as a specific element t of GF(q), provided this point arises as a point in the range of the mapping at hand. For Montgomery curves and twisted Edwards curves, this covers roughly half of the curve points; for Weierstrass curves, roughly 3/8 of the curve points. One can extend the mappings above, by mapping a pair (t1, t2) of inputs to the point Q:=P2(t1, t2):=P(t1) + P(t2). In this case, each curve point has roughly q/4 representations as an ordered pair (t1, t2) on average. In fact, one can show that if the input pairs are generated uniformly at random, then the corresponding curve points follow a distribution that is also (statistically indistinguishable from) a uniform distribution, and vice-versa. Here, each pair (t1, t2) deterministically yields a curve point, whereas for each curve point Q, a randomized algorithm yields an ordered pair (t1, t2) of pre-images of Q, where the expected number of randomized pre-images one has to try is small (four if one uses the mapping of Appendix K.3.1; two if one uses the mapping of Appendix K.3.2). For further details, see Algorithm 1 of [Tibouchi].

Similar properties hold if one uses the mappings of Appendix K.4 (rather than those of Appendix K.3): in this case, the mapping allows one to represent a curve point Q as a specific element (t,s) of GF(q)x{0,1}, provided this point arises as a point in the range of the mapping at hand. For Montgomery curves and twisted Edwards curves, this covers roughly half of the curve points; for Weierstrass curves, roughly 3/8 of the curve points. One can extend the mappings above, by mapping a pair ((t1,s1), (t2,s2)) of inputs to the point Q:=Q2((t1,s1), (t2,s2)):=Q(t1,s1) - Q(t2,s2). In this case, each curve point has roughly q representations as an ordered pair ((t1,s1), (t2,s2)) on average. In fact, one can show that if the input pairs are generated uniformly at random, then the corresponding curve points follow a distribution that is also (statistically indistinguishable from) a uniform distribution, and vice-versa. Here, each pair ((t1,s1), (t2,s2)) deterministically yields a curve point, whereas for each curve point Q, a randomized algorithm yields an ordered pair ((t1,s1), (t2,s2)) of pre-images of Q, where the expected number of randomized pre-images one has to try is small (four if one uses the mapping of Appendix K.4.1; two if one uses the mapping of Appendix K.4.2). Further details are out of scope.

NOTE 1: The main difference between the two constructions above is that the first construction uses the mappings to curve points

described in Appendix K.3, while the second construction uses the mappings to high-order curve points described in Appendix K.4. Note that Q2((t1,s1), (t2,s2)) assumes all values (+/-) P(t1) (+/-) P(t2) if one considers all possible values for the binary digits s1 and s2. (This, thereby, includes the value P2(t1, t2).)

NOTE 2: The results on the statistical distributions mentioned above still hold in practice if one makes a few localized changes to the constructions. In particular, these are independent of the specific choices for the point P0 (used with input -1 with the mappings of Appendix K.3, if applicable, respectively, used with the mappings of Appendix K.4) and also still hold if one re-defines the mappings P2 or Q2 locally so as to avoid points in the small subgroup.

K.6. Completing the Mappings to Curve Points

The mappings of Appendix K.4 operate on input pairs (t, s), where t is an element of GF(q) that is not a square in GF(q) and where s is a binary digit from the set {0,1}. One can use these mappings to produce mappings that operate on input pairs (u, s), where u is any nonzero element of GF(q), via function composition, where one first maps the pair (u,s) to the pair (t,s):=(delta*u^2,s), where delta is a fixed element of GF(q) that is not a square in GF(q), and where one subsequently applies any of forementioned mappings to the resulting pair to yield a point of the curve in question. The resulting mapping to high-order curve points can be extended further to one that operates on all elements of GF(q)x{0,1} by mapping each input (u, s) with u=0 to any fixed high-order point P1 of the curve in question. The resulting mapping is uniquely defined after fixing the curve offset P0 (used with the mappings of Appendix K.4), the high-order point P1 (used for inputs with u=0 above), and the non-square element delta of GF(q) (used for nonzero inputs u above).

For the mappings of Appendix K.3, one can use a similar function composition, where one simply drops the binary digit s and maps 0 to the point at infinity or any other suitable curve point P1. As before, the resulting mapping is uniquely defined after fixing the point P0 (for input -1 with the mappings of Appendix K.3, if applicable), the point P1 (used for input u=0 above), and the non-square element delta of GF(q) (used for nonzero inputs u above). Further details are out of scope.

Similarly, one can use the completed mappings above to map a pair ((u1,s1), (u2,s2)) of elements of GF(q)x{0,1} to a point of a curve, via function composition, where, in the first case, one first maps the pair ((u1,s1),(u2,s2)) to the pair ((t1,s1), (t2,s2)):=((delta*u1^2, s1), (delta*u2^2, s2)) and subsequently computes Q2compl((t1,s1), (t2,s2)):=Qcompl(t1,s1) - Qcompl(t2,s2),

where Qcompl(t,s):=Q(t,s) if t is nonzero and where Qcompl(0,s):=P0 otherwise (irrespective of the value of s). In the second case, one first maps the pair (u1, u2) to the pair (t1, t2):=(delta*u1^2, delta*u2^2) and subsequently computes P2compl(t1, t2):=Pcompl(t1) + Pcompl(t2), where Pcompl(t):=P(t) if t is nonzero and where Pcompl(0):=P1 otherwise. In either case, again, the resulting mapping is uniquely defined after fixing the points P0 and P1 and the non-square element delta of GF(q).

NOTE 1: Each of the above mappings is fully and unambiguously defined by the triple (P0,P1,delta). One can locally change this mapping so as to avoid points in the small subgroup, should these otherwise occur, e.g., by setting any such re-defined image to any fixed high-order point P2 of the curve in question. In this case, the corresponding mapping is uniquely defined by the quadruple (P0,P1,P2,delta) and -- in practice -- has the same statistical distribution properties as the original mapping (see NOTE 2 of Appendix K.5). For each curve in Table 1, these completed mappings are uniquely defined by the mentioned fixed curve offset P0 and non-square element delta of GF(q), if one defines P2:=P1:=P0 (henceforth called the default completed mappings).

NOTE 2: For elliptic curves defined over prime fields (i.e., q:=p) one can relax the completed mappings above and show that the statistical properties for randomized representations still hold if u1 is a random element of a sufficiently large interval in GF(p) and if u2 is a random element of a sufficiently large subset of GF(p) (see, e.g., [Tibouchi-cleancut]). This allows generating u1 and u2, e.g., each as random bit strings of length m-1, where m is the bit-length of p, thereby allowing the pair (u1, u2) -- a random (2*m-2)-bit string -- to be used unaltered in this construction, without the need to carry out a reduction modulo p first. Table 2 illustrates how this can be used to realize randomized representations and completed mappings for each curve in Table 1, where these randomized bit strings have the same byte-length as the (tight) representation of affine curve points. (Here, the field elements u1 and u2 are obtained from their bit string representations using the BS2OS mapping of Appendix I.4 and the (non-strict) OS2FE mapping of Appendix I.5.) For each curve in Table 2, we refer to this version of the default completed mapping as being the "clean-cut" default completed mapping.

Curve	left-side	right-side
NIST P-224 [FIPS-186-4]	{u1:224}	{s1:1, s2:1, u2:222}
NIST P-256 [FIPS-186-4]	{s1:1, u1:255}	{s2:1, u2:255}
NIST P-384 [FIPS-186-4]	{u1:384}	{s1:1, s2:1, u2:382}
NIST P-521 [FIPS-186-4]	{s1:1, u1:527}	{s2:1, u2:527}
brainpoolP224r1 [RFC5639]	{s1:1, u1:223}	{s2:1, u2:223}
brainpoolP256r1 [RFC5639]	{s1:1, u1:255}	{s2:1, u2:255}
brainpoolP320r1 [RFC5639]	{s1:1, u1:319}	{s2:1, u2:319}
brainpoolP384r1 [RFC5639]	{s1:1, u1:383}	{s2:1, u2:383}
brainpoolP512r1 [RFC5639]	{s1:1, u1:511}	{s2:1, u2:511}
Curve25519 [RFC7748]	{s1:1, u1:255}	{s2:1, u2:255}
Wei25519 [Appendix E.3]	{s1:1, u1:255}	{s2:1, u2:255}
Wei25519.2 [Appendix G.3]	{s1:1, u1:255}	{s2:1, u2:255}
Wei25519.-3 [Appendix G.3]	{s1:1, u1:255}	{s2:1, u2:255}
Curve448 [RFC7748]	{u1:448}	{s1:1, s2:1, u2:446}
Wei448 [Appendix M.3]	{u1:448}	{s1:1, s2:1, u2:446}
Wei448.1 [Appendix N.3]	{u1:448}	{s1:1, s2:1, u2:446}
Wei448.-3 [Appendix N.3]	{u1:448}	{s1:1, s2:1, u2:446}
secp256k1.m [Appendix L.3]	{u1:256}	{s1:1, s2:1, u2:254}

Table 2: Randomized representation of curve points, for some curves of practical interest, including curve-specific relative ordering and bit-length of substrings representing the tuple ((u1,s1),(u2,s2)), resulting in the bit string left-side || right-side. (Tailored towards avoiding modular reductions in mappings to curve points.)

Table 3 shows an alternative arrangement, tailored towards optimizing the efficiency of computing randomized representations of curve points (see Appendix K.5), rather than towards avoiding modular reductions in the mappings to curve points. (Here, we used randomized representations of elements of GF(p), when appropriate, and the bias upper bound 2^{-64} from Table 4.) For each curve in Table 3, we refer to this version of the default completed mapping as being the "point-randomization-optimized" default completed mapping (where both versions coincide if the prime number p is relatively close to a power of two). (Here, the field elements u1 and u2 are obtained from their bit string representations using the BS2OS

mapping of Appendix I.4 and the (non-strict) OS2FE mapping of Appendix I.5.) Suitability of each of these completed mappings is application-specific (and also depends on the maximum bias one can tolerate). Further details are out of scope of this document.

Curve	left-side	right-side
NIST P-224 [FIPS-186-4]	{u1:224}	{s1:1, s2:1, u2:222}
NIST P-256 [FIPS-186-4]	{u1:288}	{s1:1, s2:1, u2:222}
NIST P-384 [FIPS-186-4]	{u1:384}	{s1:1, s2:1, u2:382}
NIST P-521 [FIPS-186-4]	{s1:1, u1:527}	{s2:1, u2:527}
brainpoolP224r1 [RFC5639]	{s1:1, u1:287}	{s2:1, u2:159}
brainpoolP256r1 [RFC5639]	{s1:1, u1:319}	{s2:1, u2:191}
brainpoolP320r1 [RFC5639]	{s1:1, u1:383}	{s2:1, u2:255}
brainpoolP384r1 [RFC5639]	{s1:1, u1:447}	{s2:1, u2:319}
brainpoolP512r1 [RFC5639]	{s1:1, u1:575}	{s2:1, u2:447}
Curve25519 [RFC7748]	{s1:1, u1:255}	{s2:1, u2:255}
Wei25519 [Appendix E.3]	{s1:1, u1:255}	{s2:1, u2:255}
Wei25519.2 [Appendix G.3]	{s1:1, u1:255}	{s2:1, u2:255}
Wei25519.-3 [Appendix G.3]	{s1:1, u1:255}	{s2:1, u2:255}
Curve448 [RFC7748]	{u1:448}	{s1:1, s2:1, u2:446}
Wei448 [Appendix M.3]	{u1:448}	{s1:1, s2:1, u2:446}
Wei448.1 [Appendix N.3]	{u1:448}	{s1:1, s2:1, u2:446}
Wei448.-3 [Appendix N.3]	{u1:448}	{s1:1, s2:1, u2:446}
secp256k1.m [Appendix L.3]	{u1:256}	{s1:1, s2:1, u2:254}

Table 3: Randomized representation of curve points, for some curves of practical interest, including curve-specific relative ordering and bit-length of substrings representing the tuple ((u1,s1),(u2,s2)), resulting in the bit string left-side || right-side. (Tailored towards efficient computation of randomized representations of curve points.)

Appendix L. Curve secp256k1 and Friend

This section illustrates how isogenies can be used to yield curves with specific properties (here, illustrated for the "BitCoin" curve secp256k1).

L.1. Curve Definition and Alternative Representation

The elliptic curve secp256k1 is the Weierstrass curve W_{a,b} defined over the prime field GF(p), with p:=2^256-2^32-2^9-2^8-2^7-2^6-2^4-1, where a:=0 and b:=7. This curve has order h*n, where h=1 and where n is a prime number. For this curve, domain parameter a is zero, whereas b is not. The quadratic twist of this curve has order h1*n1, where h1 is a 37-bit integer and where n1 is a prime number. For this curve, the base point is the point (GX, GY).

The curve secp256k1 is 3-isogenous to the Weierstrass curve secp256k1.m defined over GF(p), which has nonzero domain parameters a and b and has as base point the pair (GmX,GmY), where parameters are as specified in Appendix L.3 and where the related mappings are as specified in Appendix L.2.

L.2. Switching Between Representations

Each affine point (X,Y) of secp256k1 corresponds to the point (X',Y'):=(u(X)/w(X)^2,Y*v(X)/w(X)^3) of secp256k1.m, where u, v, and w are the polynomials with coefficients in GF(p) as defined in Appendix L.4.1, while the point at infinity of secp256k1 corresponds to the point at infinity of secp256k1.m. Under this isogenous mapping, the base point (GX,GY) of secp256k1 corresponds to the base point (GmX,GmY) of secp256k1.m. The dual isogeny maps the affine point (X',Y') of secp256k1.m to the affine point (X,Y):=(u'(X')/w'(X')^2,Y'*v'(X')/w'(X')^3) of secp256k1, where u', v', and w' are the polynomials with coefficients in GF(p) as defined in Appendix L.4.2, while mapping the point at infinity O of secp256k1.m to the point at infinity O of secp256k1. Under this dual isogenous mapping, the base point (GmX, GmY) of secp256k1.m corresponds to a multiple of the base point (GX, GY) of secp256k1, where this multiple is l=3 (the degree of the isogeny; see the description in Appendix F.4). Note that this isogenous map (and its dual) primarily involves the evaluation of three fixed polynomials involving the x-coordinate, which takes roughly 10 modular multiplications (or less than 1% relative incremental cost compared to the cost of an elliptic curve scalar multiplication).

L.3. Domain Parameters

The parameters of the curve sec256k1 and the corresponding 3-isogenous curve sec256k1.m are as indicated below. Here, the domain parameters of the curve secp256k1 are as specified in [SEC2]; the domain parameters of secp256k1.m are "new".

General parameters (for all curves):

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

(=0xffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe fffffc2f)

h 1

n 115792089237316195423570985008687907852837564279074904382605163141518161494337

(=0xffffffff ffffffff ffffffff fffffffe baaedce6 af48a03b bfd25e8c d0364141)

h1 23479460174521 (=0x1a 9bfcab89)

n1 10131766773001318469008702396060356387381009972480920692566974370 31

(=0x099ee564 ea5d84f5 08913936 a761b0d5 d792a426 a7779817 ae2f5b67)

Weierstrass curve-specific parameters (for secp256k1):

a 0 (=0x00)

b 7 (=0x07)

GX 55066263022277343669578718895168534326250603453777594175500187360389116729240

(=0x79be667e f9dcbbac 55a06295 ce870b07 029bfcdb 2dce28d9 59f2815b 16f81798)

GY 32670510020758816978083085130507043184471273380659243275938904335757337482424

(=0x483ada77 26a3c465 5da4fbfc 0e1108a8 fd17b448 a6855419 9c47d08f fb10d4b8)

Weierstrass curve-specific parameters (for secp256k1.m):

a 93991599167772749909245591943117186381494883464374162770646538702960816911535

(=0xcfcd5c21 75e2ef7d ccdce737 770b7381 5a2f13c5 09035ca2 54a14ac9 f08974af)

b 1771 (=0x06eb)

GmX 26591621185618668069038227574782692264471832498547635565821216767730887659845

(=0x3aca5300 959fa1d0 baf78dcf f77a616f 395e586d 67aced0a 88798129 0c279145)

GmY 67622516283223102233819216063319565850973524550533340939716651159860372686848

(=0x9580fce5 3a170f4f b744579f f3d62086 12cd6a23 3e2de237 f976c6a7 8611c800)

L.4. Isogeny Details

The isogeny and dual isogeny are both isogenies with degree l=3. Both are specified by a triple of polynomials u, v, and w (resp. u', v', and w') of degree 3, 3, and 1, respectively, with coefficients in GF(p). The coeffients of each of these polynomials are specified in Appendix L.4.1 (for the isogeny) and in Appendix L.4.2 (for the dual isogeny). For each polynomial in variable x, the coefficients are tabulated as the sequence of coefficients of x^0, x^1, x^2, ..., in hexadecimal format.

L.4.1. Isogeny Parameters

L.4.1.1. Coefficients of u(x)

0 0x54

1 0xa4d89db3ed06c81e6143ec2eca9f761d8d17260dc229e1da1f73f714506872a9

2 0xcc58ffccbd9febb4a66222c7d1311d988d88c0624bcd68ec4c758a8e67dfd99b

3 0x01

L.4.1.2. Coefficients of v(x)

0 0x1c

1 0x94c7bc69befd17f2fae2e3ebf24df1f355d181fa1a8056103ba9baad4b40f029

2 0xb2857fb31c6fe18ef993342bb9c9ac64d44d209371b41d6272b04fd61bcfc851

3 0x01

L.4.1.3. Coefficients of w(x)

0 0xe62c7fe65ecff5da53311163e8988ecc46c4603125e6b476263ac546b3efeae5

1 0x01

L.4.2. Dual Isogeny Parameters

L.4.2.1. Coefficients of u'(x)

0 0x8e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38daaaaa8c7

1 0x44cd5cd7ce55a801725891578fbe7356bd936355fd0e2f538797cecff7a37244

2 0x668d0011162006c3c889f4680f9a4b77d0d26a89e6bb87b13bd8d1cfdd600a41

3 0x8e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38daaaaa88c

L.4.2.2. Coefficients of v'(x)

0 0x4bda12f684bda12f684bda12f684bda12f684bda12f684bda12f684b8e38e23c

1 0x519ba9c1f48f68054def6a410f0fa6e8b71c6c3b4a8958324681f6508c01fada

2 0xb34680088b100361e444fa3407cd25bbe8693544f35dc3d89dec68e76eb00338

3 0x2f684bda12f684bda12f684bda12f684bda12f684bda12f684bda12f38e38d84

L.4.2.3. Coefficients of w'(x)

0 0x4d7a804ce3901e71066ccbd44636539b2bb2df6c8e4be29d8d4fb028e43033de

1 0x01

Appendix M. Curve448 and Cousins

This section introduces curves related to Curve448 and explains their relationships.

M.1. Curve Definition and Alternative Representations

The elliptic curve Curve448 is the Montgomery curve M_{A,B} defined over the prime field GF(p), with p:=2^{448}-2^{224}-1, where A:=156326 and B:=1. This curve has order h*n, where h=4 and where n is a prime number. For this curve, A^2-4 is not a square in GF(p), whereas A-2 is. The quadratic twist of this curve has order h1*n1, where h1=4 and where n1 is a prime number. For this curve, the base

point is the point (Gu, Gv), where Gu=5 and where Gv is an even integer in the interval [0, p-1].

This curve has the same group structure as (is "isomorphic" to) the twisted Edwards curve E_{a,d} defined over GF(p), with as base point the point (Gx, Gy), where parameters are as specified in Appendix M.3. This curve is denoted as Ed448. For this curve, the parameter a is a square in GF(p), whereas d is not, so the group laws of Appendix C.3 apply.

The curve is also isomorphic to the elliptic curve W_{a,b} in short-Weierstrass form defined over GF(p), with as base point the point (GX, GY), where parameters are as specified in Appendix M.3. This curve is denoted as Wei448.

M.2. Switching between Alternative Representations

Each affine point (u, v) of Curve448 corresponds to the point (X, Y):=(u + A/3, v) of Wei448, while the point at infinity of Curve448 corresponds to the point at infinity of Wei448. (Here, we used the mappings of Appendix D.2 and that B=1.) Under this mapping, the base point (Gu, Gv) of Curve448 corresponds to the base point (GX, GY) of Wei448. The inverse mapping maps the affine point (X, Y) of Wei448 to (u, v):=(X - A/3, Y) of Curve448, while mapping the point at infinity of Wei448 to the point at infinity of Curve448. Note that this mapping involves a simple shift of the first coordinate and can be implemented via integer-only arithmetic as a shift of -delta for the isomorphic mapping and a shift of delta for its inverse, where delta:=(p-A)/3 is the integer defined by

```
delta  2422795747652022968497746026293348447845471202291060200938300663935374427222435908954654612328921819766962948206145457870178326
    72736371

    (=0x55555555 55555555 55555555 55555555 55555555 55555555
    55555554 ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
    ffff3473).
```

(Note that, depending on the implementation details of the field arithmetic, one may have to shift the result by +p or -p if this integer is not in the interval [0,p-1].)

The curve Ed448 is isomorphic to the curve Curve448, where the base point (Gu, Gv) of Curve448 corresponds to the base point (Gx,Gy) of Ed448 and where the point at infinity and the point (0,0) of order two of Curve448 correspond to, respectively, the point (0, 1) and the point (0, -1) of order two of Ed448 and where each other point (u, v)

of Curve448 corresponds to the point (c*u/v, (u+1)/(u-1)) of Ed448, where c is the element of GF(p) defined by

c sqrt((A-2)/B)

197888467295464439538354009753858038256835152591059802148199779196087404232002515713604263127793030747855424464185691766453844835192428

(=0x45b2c5f7 d649eed0 77ed1ae4 5f44d541 43e34f71 4b71aa96 c945af01 2d182975 0734cde9 faddbda4 c066f7ed 54419ca5 2c85de1e 8aae4e6c).

(Here, we used the mapping of Appendix D.1 and normalized this using the mapping of Appendix F.1 (where the element s of that appendix is set to c above).) The inverse mapping from Ed448 to Curve448 is defined by mapping the point (0, 1) and the point (0, -1) of order two of Ed448 to, respectively, the point at infinity and the point (0,0) of order two of Curve448 and having each other point (x, y) of Ed448 correspond to the point ((y + 1)/(y - 1), c*(y + 1)/((y-1)*x)) of Curve448.

The curve Ed448 is isomorphic to the Weierstrass curve Wei448, where the base point (Gx, Gy) of Ed448 corresponds to the base point (GX,GY) of Wei448 and where the identity element (0,1) and the point (0,-1) of order two of Ed448 correspond to, respectively, the point at infinity O and the point (A/3, 0) of order two of Wei448 and where each other point (x, y) of Ed448 corresponds to the point (X, Y):=((y+1)/(y-1)+A/3, c*(y+1)/((y-1)*x)) of Wei448, where c was defined before. (Here, we used the mapping of Appendix D.3.) The inverse mapping from Wei448 to Ed448 is defined by mapping the point at infinity O and the point (A/3, 0) of order two of Wei448 to, respectively, the identity element (0,1) and the point (0,-1) of order two of Ed448 and having each other point (X, Y) of Wei448 correspond to the point (c*(X-A/3)/Y, (X-A/3+1)/(X-A/3-1)) of Ed448.

Note that these mappings can be easily realized if points are represented in projective coordinates, using a few field multiplications only, thus allowing switching between alternative curve representations with negligible relative incremental cost.

M.3. Domain Parameters

The parameters of the Montgomery curve and the corresponding isomorphic curves in twisted Edwards curve and short-Weierstrass form are as indicated below. Here, the domain parameters of the Montgomery curve Curve448 and of the twisted Edwards curve Ed448 are as specified in [RFC7748]; the domain parameters of Wei448 are "new".

IMPORTANT NOTE: the supposed base point of Ed448 specified in [RFC7748] is incorrect, since it has order 2*n, and - in the notation below - that point is the point (Gx,-Gy)=-(Gx, Gy)+(0,-1). The birational map in that document is also incorrect.

General parameters (for all curve models):

p 2^{448}-2^{224}-1

(=0xffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff)

h 4

n 181709681073901722637330951972001133588410340171829515070372549795146003961539585716195755291692375963310293709091662304773755859649779

(=2^{446} - 0x8335dc16 3bb124b6 5129c96f de933d8d 723a70aa dc873d6d 54a7bb0d)

h1 4

n1 181709681073901722637330951972001133588410340171829515070372549795160160121825800627002436557645897001734148521830156375752993149532941

(=2^{446} + 0x0335dc16 3bb124b6 5129c96f de933d8d 723a70aa dc873d6d 54a7bb0d)

Montgomery curve-specific parameters (for Curve448):

A 156326 (=0x0262a6)

B 1 (=0x01)

Gu 5 (=0x05)

Gv 355293926785568175264127502063783334808976399387714271831880898435169088786967410002932673765864550910142774147268105838985595290606362

(=0x7d235d12 95f5b1f6 6c98ab6e 58326fce cbae5d34 f55545d0 60f75dc2 8df3f6ed b8027e23 46430d21 1312c4b1 50677af7 6fd7223d 457b5b1a)

Edwards curve-specific parameters (for Ed448):

a 1 (0x01)

d 39082/39081 = (A+2)/(A-2)

(=611975850744529176160423220965553317543219696871016626328968936
4150878600426364748917855992836660204147686789799893781470654628 1
5545017)

(=0xd78b4bdc 7f0daf19 f24f38c2 9373a2cc ad461572 42a50f37
809b1da3 412a12e7 9ccc9c81 264cfe9a d0809970 58fb61c4 243cc32d
baa156b9)

Gx 345397493039729516374008604150537410266655260075183290216406970 28
164569507367234443048178775934063322170839158342404178892412456 77
00732

(=0x79a70b2b 70400553 ae7c9df4 16c792c6 1128751a c9296924
0c25a07d 728bdc93 e21f7787 ed697224 9de732f3 8496cd11 69871309
3e9c04fc)

Gy 3/2

363419362147803445274661903944002267176820680343659030140745099 59
030616408336538634319819184933827296504444223092181868052674900 91
82721

(=0x7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff 80000000 00000000 00000000 00000000 00000000 00000000
00000001)

Weierstrass curve-specific parameters (for Wei448):

a 484559149530404593699549205258669689569094240458212040018766013278
707488544448718179093092246578436395339258964122909157403565719 96
37535

(=0xaaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
aaaaaaa9 ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe
1a76d41f)

b 269199527516891440944194002921483160871719022476784466770922295 99
281938080249287877273940136988020219632921646734949531919168566 45
13904

(=0x5ed097b4 25ed097b 425ed097 b425ed09 7b425ed0 97b425ed
097b425e 71c71c71 c71c71c7 1c71c71c 71c71c71 c71c71c7 1c72c87b
7cc69f70)

GX 484559149530404593699549205258669689569094240458212040187660132787074885444487181790930922465784363953392589641229091574035665345629073

(=0xaaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa 00000000 00000000 00000000 00000000 00000000 00000000 0000cb91)

GY 355293926785568175264127502063783334808976399387714271831880898435169088786967410002932673765864550910142774147268105838985595290606362

(=0x7d235d12 95f5b1f6 6c98ab6e 58326fce cbae5d34 f55545d0 60f75dc2 8df3f6ed b8027e23 46430d21 1312c4b1 50677af7 6fd7223d 457b5b1a)

Appendix N. Further Cousins of Curve448

This section introduces some further curves related to Curve448 and explains their relationships.

N.1. Further Alternative Representations

The Weierstrass curve Wei448 is isomorphic to the Weierstrass curve Wei448.1 defined over GF(p), with as base point the pair (G1X,G1Y), and isogenous to the Weierstrass curve Wei448.-3 defined over GF(p), with as base point the pair (G3X, G3Y), where parameters are as specified in Appendix N.3 and where the related mappings are as specified in Appendix N.2.

The Edwards curve Ed448 is isogenous to the Edwards curve Edwards448 defined over GF(p), with as base point the pair (G1x,G1y), where parameters are as specified in Appendix N.3 and where the related mappings are as specified in Appendix N.2. For this curve, the domain parameter a is a square in GF(p), whereas d1 is not, so the group laws of Appendix C.3 apply.

N.2. Further Switching

Each affine point (X, Y) of Wei448 corresponds to the point (X', Y'):=(X*s^2,Y*s^3) of Wei448.1, where s is the element of GF(p) defined by

s 523222743436774427793795205897710288185684045877291179195905110619350951023834788013447388868747146521664184623272464129895489080000881

(=0xb848cd01 981d2f83 f2829b42 eb86914e 88f44c9d 05dcbdff dbdd1e56 c4674bc8 d6d90d91 862a38f5 ca797ca7 f21c05cf a7ac32bf d2ca0171),

while the point at infinity of Wei448 corresponds to the point at infinity of Wei448.1. (Here, we used the mapping of Appendix F.3.) Under this mapping, the base point (GX, GY) of Wei448 corresponds to the base point (G1X,G1Y) of Wei448.1. The inverse mapping maps the affine point (X', Y') of Wei448.1 to (X,Y):=(X'/s^2,Y'/s^3) of Wei448, while mapping the point at infinity O of Wei448.1 to the point at infinity O of Wei448. Note that this mapping (and its inverse) involves a modular multiplication of both coordinates with fixed constants s^2 and s^3 (respectively, 1/s^2 and 1/s^3), which can be precomputed.

The point at infinity and the point (A/3,0) of order two of Wei448 both correspond to the point at infinity of Wei448.-3, while each other point (X,Y) of Wei448 corresponds to the point (X',Y'):=(X1*t^2,Y1*t^3) of Wei448.-3, where (X1,Y1)=(u(X)/w(X),Y*v(X)/w(X)^2), where u, v, and w are the polynomials with coefficients in GF(p) as defined in Appendix N.4.1 and where t is the element of GF(p) defined by

t 23579450751475691430882365546539966269774125426758968522698856022 13378944265540874438945283200254318223329383397068961863760712339 07365

(=0x530c9a1d 7cf071d0 9646b83d b246626b 4e57ba5d 6a791bef 76197254 3209dc5c 20d81498 d5ab8d7a 2fb22507 ca68c040 a6c82eb3 b6c7aaa5).

(Here, we used the isogenous mapping of Appendix F.4.) Under this isogenous mapping, the base point (GX,GY) of Wei448 corresponds to the base point (G3X,G3Y) of Wei448.-3. The dual isogeny maps the point at infinity O and the point (tau,0) of order two of Wei448.-3, where tau is the element of GF(p) defined by

tau 42178595713080601145580616893463205889346047807394283240821661315 01870168726890624132409486822657385666418069563147259152341712826 86207

(=0x948eabcf 057e0d55 9c372c98 075ddacf 6f3d19bc 514e5d23 248d685b 75f97a10 36696aaf 61c02d8e 3da778c3 8d9fda05 54c9258b 3c0e80ff),

to the point at infinity O of Wei448, while mapping each other point (X',Y') of Wei448.-3 to the affine point (X,Y):=(u'(X1)/w'(X1),Y1*v'(X1)/w'(X1)^2) of Wei448, where

(X1,Y1)=(X'/t^2,Y'/t^3) and where u', v', and w' are the polynomials with coefficients in GF(p) as defined in Appendix N.4.2. Under this dual isogenous mapping, the base point (G3X, G3Y) of Wei448.-3 corresponds to a multiple of the base point (GX, GY) of Wei448, where this multiple is l=2 (the degree of the isogeny; see the description in Appendix F.4). Note that this isogenous map (and its dual) primarily involves the evaluation of three fixed polynomials involving the x-coordinate, which takes only a few modular multiplications (less than 0.5% relative incremental cost compared to the cost of an elliptic curve scalar multiplication).

Each point (x1,y1) of Edwards448 with nonzero coordinates corresponds to the point (x,y) of Ed448, where

x = c*x1*y1/(1-d1*x1^2*y1^2) = c*x1*y1/(2-x1^2-y1^2) and

y =(1 + d1*x1^2*y1^2)/(y1^2-x1^2) = -(x1^2+y1^2)/(x1^2-y1^2),

while each other point (i.e., a point of order 1, 2, or 4) corresponds to the identity element (0,1) of Ed448. (Here, we used the 4-isogenous mapping of Appendix F.4). Under this isogenous mapping, the base point (G1x, G1y) of Edwards448 corresponds to the base point (Gx,Gy) of Ed448. The dual isogeny maps each point (x,y) of Ed448 to the point (x1,y1) of Edwards448, where

x1 = (4*x*y/c)/(y^2-x^2) and

y1 = (1 - d*x^2*y^2)/(1 + d*x^2*y^2) = (2-x^2-y^2)/(x^2+y^2).

Under this dual isogenous mapping, the base point (Gx, Gy) of Ed448 corresponds to a multiple of the base point (G1x, G1y) of Edwards448, where this multiple is l=4 (the degree of the isogeny; see the description in Appendix F.4). Note that this isogenous map (and its dual) primarily involves the evaluation of three fixed polynomials, which takes only a few multiplications (less than 0.5% relative incremental cost compared to the cost of an elliptic curve scalar multiplication).

Each point (x1,y1) of Edwards448 with nonzero coordinates corresponds to the point (u,v) of Curve448, where

u = y1^2/x1^2 and v = y1*(2-x1^2-y1^2)/x1^3,

while each other point (i.e., a point of order 1, 2, or 4) corresponds to the point at infinity of Curve448. Under this isogenous mapping, the base point (G1x, G1y) of Edwards448 corresponds to the base point (Gu,Gv) of Curve448. The dual isogeny maps both the point at infinity and the point (0,0) of order two of

Curve448 to the identity element (0,1) of Edwards448, while each other point (u,v) of Curve448 corresponds to the point (x1,y1) of Edwards448, where

x1 = 4*(u^2-1)*v/((u^2-1)^2+4*v^2) and

y1 = u*((u^2-1)^2-4*v^2)/(2*(u^2+1)*v^2-u*(u^2-1)^2).

Under this dual isogenous mapping, the base point (Gu, Gv) of Curve448 corresponds to a multiple of the base point (G1x, G1y) of Edwards448, where this multiple is l=4 (the degree of the isogeny; see above).

N.3. Further Domain Parameters

The parameters of the Weierstrass curve with a=1 that is isomorphic with Wei448 and the parameters of the Weierstrass curve with a=-3 that is isogenous with Wei448 are as indicated below. Both domain parameter sets can be exploited directly to derive more efficient point addition formulae, should an implementation facilitate this. The domain parameters of the Edwards curve Edwards448 are as specified in [RFC7748].

General parameters: same as for Wei448 (see Appendix M.3)

Weierstrass curve-specific parameters (for Wei448.1, i.e., with a=1):

a 1 (=0x01)

b 659612817018071705319448049859079902872252480565600363923809459513818308850763543778602104492771511922449740791489579066934526889652743

(=0xe8528596 bfbcbac9 7ebdbe4e 9683e25c 73a5ff37 6c4cd400 5a75c425 8e3eb05a 9f6f8c24 24cb5aa9 0dcf9fa4 cab6691d 5530347c 28437207)

G1X 192362119825082116448050334593062730385232304813091415185404141637209118629245848223191246024325724747868400544899974680969100799597723

(=0x06c672d5 b5bae33b 010fa210 9de7937a 95db8ffc 043c507f 5e0d07a1 25382eaf 13f5fc3b 75db2614 6e6d002f d8364ed6 c9bc8fbf bbda22ab)

G1Y 303194430568771698044880723845630642886755762341967736679208075677917792785875562195875622220663246598830846631955694882177584586164158

(=0x6ac9c53c 767cd3ae cbf904a1 2923502f 115355d1 6ae8911c
5c92f612 aa854455 d1e6d29f 4db4ddea 519a174f c0dd2505 ec3328ba
250a07be)

Weierstrass curve-specific parameters (for Wei448.-3, i.e., with a=-3):

a -3

(=0xffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
fffffffe ffffffff ffffffff ffffffff ffffffff ffffffff
fffffffc)

b 699937686810001500848336699619005330673833355924944987095346934649131425073158306877468995089322968102492731574779458733142208859254465

(=0xf686723d 80e29d06 2d00a9f1 3305b698 85790019 cca78035
9dac226b efb1ae21 125397dd 16f255b0 cc5d18e5 43582a1c af90dfe2
c0aeaec1)

G3X 406774749948698764709161334243115168566624079707994248378413484218769627466511314071900122703011655137887728036852633498562710468088795

(=0x8f452c6b dc3265dd 580b2638 59a02b20 198cc020 1dd7fba1
8b431694 4a936052 fb4e4a41 93d01fa5 5fb5c732 7393208b 8170f3f2
be78d3db)

G3Y 54594210970205994927260789585006437115117066846498189378285031510903102904683477149293661066354709786667955124466290512357045048680614 7

(=0xc0494f90 461db11c 35fb7646 8349399a ae230351 11330cce
b7473244 ab63c955 cf6ec02f 2656b439 44b19f4b 52eef12e 73026bbc
84444683)

Edwards curve-specific parameters (for Edwards448):

a 1 (0x01)

d1 -39081 = -(A-2)/4

(=726838724295606890549323807888004534353641360687318060281490199180612328166730772686396383698676545930088884461843637361053498018326358)

(=0xffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
fffffffe ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffff6756)

G1x 22458004029592430018760433409989603624678964163256413424612546168
69504154674060329090291928693579532825780320751464461736746026352
47710

(=0x4f1970c6 6bed0ded 221d15a6 22bf36da 9e146570 470f1767
ea6de324 a3d3a464 12ae1af7 2ab66511 433b80e1 8b00938e 2626a82b
c70cc05e)

G1y 29881921007848149267601793044393067343754404015408024209592824137
23315061898358760035368786554187847339823032335034625005315450628
32660

(=0x693f4671 6eb6bc24 88762037 56c9c762 4bea7373 6ca39840
87789c1e 05a0c2d7 3ad3ff1c e67c39c4 fdbd132c 4ed7c8ad 9808795b
f230fa14)

N.4. Isogeny Details

The isogeny and dual isogeny are both isogenies with degree l=2. Both are specified by a triple of polynomials u, v, and w (resp. u', v', and w') of degree 2, 2, and 1, respectively, with coefficients in GF(p). The coeffients of each of these polynomials are specified in Appendix N.4.1 (for the isogeny) and in Appendix N.4.2 (for the dual isogeny). For each polynomial in variable x, the coefficients are tabulated as the sequence of coefficients of x^0, x^1, x^2, ..., in hexadecimal format.

N.4.1. Isogeny Parameters

N.4.1.1. Coefficients of u(x)

0 0x01

1 0x55555555555555555555555555555555555555555555555555555554ffffffff
ffffffffffffffffffffffffffffffffffffffff3473

2 0x01

N.4.1.2. Coefficients of v(x)

0 0x1c71c71c71c71c71c71c71c71c71c71c71c71c71c71c71c71c71c55555555
55555555555555555555555555555555f72db94a

1 0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa9fffffff
ffffffffffffffffffffffffffffffffffffffffffffe68e6

2 0x01

N.4.1.3. Coefficients of w(x)

0 0x555555555555555555555555555555555555555555555555555554ffffffff
fffffffffffffffffffffffffffffffffffff3473

1 0x01

N.4.2. Dual Isogeny Parameters

N.4.2.1. Coefficients of u'(x)

0 0x016c26e0e8

1 0x555555555555555555555555555555555555555555555555555500000000
00000000000000000000000000000000000000000065c6

2 0x3fffffffffffffffffffffffffffffffffffffffffffffffffffffc0000000
00000000000000000000000000000000000000000000

N.4.2.2. Coefficients of v'(x)

0 0x8e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38e38daaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa45836c31

1 0x555555555555555555555555555555555555555555555555555500000000
00000000000000000000000000000000000000000065c6

2 0x1ffffffffffffffffffffffffffffffffffffffffffffffffffffffe0000000
00000000000000000000000000000000000000000000

N.4.2.3. Coefficients of w'(x)

0 0x555555555555555555555555555555555555555555555555555500000000
00000000000000000000000000000000000000000019719

1 0x01

Appendix O. Representation Examples Curve448 Family Members

We present some examples of computations using the curves introduced in Appendix M and Appendix N of this document. In each case, we indicate the values of P, k*P, and (k+1)*P, where P is a fixed

multiple (here: 2019) of the base point of the curve in question and where the private key k is the integer

k 626620393045239066897881248332893844462029464744400576551607736956375634250541040216623001862006648279408086664161693201332762357901952

(=0xdcb3bbb9 e42d7aca fe62052d 902123c7 0872b984 4c1e199f 7c5d37bd 1171102b c20a6352 d9c91886 29b685de 51441e84 3afe2665 5251aa80).

In the examples below, each curve point is represented using the compressed point representation (see Appendix I.8), but where (for historical reasons) the parity bit t of the compressed curve point in question is represented by 0x00 or 0x80 depending on whether t=0 or t=1, respectively. Notice that this representation corresponds to the compressed point representation of Appendix I.8, but with the bit-ordering in the 1-octet representation of t reversed. (Note that this puts the parity bit t in the leftmost bit position of the octet string if one follows the MSB/msb representation conventions.) For simplicity, this representation is again called the "squeezed" representation, although each point is now represented as a 57-octet string and, thereby, one octet longer than the tight representation of elements of GF(p). As before, the ordering convention (see Appendix I.7) depends on the underlying curve model in question. Here, points of a Weierstrass curve are represented in tight MSB/msb-order, points of a Montgomery curve in tight LSB/msb-order, and points of a twisted Edwards curve in tight LSB/lsb-order. For points that are a public key, the corresponding private keys are represented as 56-octet strings, using the same ordering conventions as with the public keys. For affine points, we also give the tight representation of each of its coordinates (as 56-octet strings), using the same ordering conventions as used with the squeezed point representation. For further details, see the examples themselves.

O.1. Example with Curve448

Pm=(u, v), k*Pm=(u1, v1), and (k+1)*Pm=(u2, v2) with Curve448:

u 532985947382990857723735360801334832366737825788953396767851799239076429830009010270945386605469506108274624363604511075029644493227715

(=0xbbb91ba3 b0ef74c3 214394b4 d8f0d32d c4a92193 5f573009 39fd86a3 8d54be2a 4d63380b 692381bb ed7339fd dca7b0cd a80166fe 18c086c3).

v 30578727850066757341435137807347775064915058999485530015946871157
867946314072749368706185807141079316617309993502226448947292856049
97149

(=0x6bb38e82 8d52337f 6f0395ef dc16c776 52162f5e 309112ae
fc7401bf 0cfb0499 eb1ed555 bf507ebc c33b4753 2d6dc6c5 d68dea1c
c1e4c1fd).

u1 64579461799301726935877447646800238443923683299745374127971411973
125152951617918897432280492222791889683658771641880750950744188068
2513

(=0xe37497bf 9f704689 54ec6537 cbbe91d0 3ffcdcdb 8b707253
a2212cdb e020ba9a 0bf65a1d 5d9a128a f85c63a2 79a00139 7aca56db
15335011).

v1 55735504615964066386264989698774850924544182484936624265048483231
356938593626278801845862824392346027980235940546117374126675437581
1547

(=0xc44e5e0f 2c254d23 1dc082db 77175e8c fd37793c 22ebe200
77905a5f 750b3c9f 4a95d4d5 4e1a1e54 d2d31689 4249252d 0c8b1c45
1c1481db).

u2 32685564331119553171673802371596819258307818641496728161547328225
075956185873236192567695585356306249605752126446801490346610082548
876

(=0x0b831eca 9c6215b0 5d830361 4013732f 7a9dd07f ebb9441e
49129264 eb724f44 dc53671c ffabb9ee 0c02aa74 b083cd82 a821a4cf
6f6d8c8c).

v2 57682103223585233918507344950062950306770296215271320612937204938
774992821034830929905101369014157572730827196656574842943443335912
0741

(=0xcb2988a4 6e37f9a9 a7a1255b 2fd2eea9 82308e7c eb8e18b8
2252175f fd416a10 5984c6b8 36470e48 31879293 8f6139c6 f96164cb
14010965).

As suggested in Appendix C.2, the v-coordinate of k*Pm can be indirectly computed from the u-coordinates of Pm, k*Pm, and (k+1)*Pm, and the v-coordinate of Pm, which allows computation of the entire point k*Pm (and not just its u-coordinate) if k*Pm is computed using the Montgomery ladder (as, e.g., [RFC7748] recommends), since that algorithm computes both u1 and u2 and the v-coordinate of the point Pm may be available from context.

The representation of k and the compressed representations of Pm and k*Pm in tight LSB/msb-order are given by

repr(k) 0x80aa5152 6526fe3a 841e4451 de85b629 8618c9d9 52630ac2
2b107111 bd375d7c 9f191e4c 84b97208 c7232190 2d0562fe
ca7a2de4 b9bbb3dc;

repr(Pm) 0xc386c018 fe6601a8 cdb0a7dc fd3973ed bb812369 0b38634d
2abe548d a386fd39 0930575f 9321a9c4 2dd3f0d8 b4944321
c374efb0 a31bb9bb 80;

repr(k*Pm) 0x11503315 db56ca7a 3901a079 a2635cf8 8a129a5d 1d5af60b
9aba20e0 db2c21a2 5372708b dbdcfc3f d091becb 3765ec54
8946709f bf9774e3 80,

where the leftmost bit of the rightmost octet indicates the parity of the v-coordinate of the point of Curve448 in question (which, in this case, are both one, since v and v1 are odd). See Appendix H.2 and Appendix I for further detail on (squeezed) point compression.

The scalar representation and (squeezed) point representation illustrated above are consistent with the representations specified in [RFC7748], except that in [RFC7748] only an affine point's u-coordinate is represented (i.e., the v-coordinate of any point is always implicitly assumed to have an even value) and that the representation of the point at infinity is not specified. (Note that due to the bit-length of the prime p, the lossless representation requires an additional octet compared to the lossy representation without v-coordinate.) Another difference is that [RFC7748] allows non-unique representations of some elements of GF(p), whereas our representation conventions do not (since tight).

A randomized representation (t1, t2) of the point k*Pm in tight LSB/msb order is given by

t1 642695971489808425948939115432957219707501931105169269237
122551860533279049805112466411050091592893048844749561382
909707113070546618079

(=0xdf86cb83 ae1ca6e6 da6afbaf afbb2fc0 606a136f 80eea078
c868a5d7 7e638d09 99518385 65250cf1 9c034f96 1fa28f54
f3016600 68335de2)

t2 569275737967591640709387827593956375775147481657775744720
460881642951497067363381071471046477130052706607411985560
522861593611384288817

(=0x3176361c 580a7bcd d7880d84 aba10bc6 57010328 afb728cc
2016461b 246bef46 0eb4bb04 8c1a3616 c3f74a56 3cc1790f
6472256b ca3481c8),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.2 with the default square root function.

This representation can also be expressed in tight LSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(0,1) and where

u1 13624339918182778128824356684066430978093755373447629798655579421282677482169738461260353906896396166856092397511742954801244490808118 1

(=0x1d50f12f 9d4a9f5b c49d8a59 0403a454 e9ab4208 ccde0595
11d72af1 f44cefe4 0c743579 6502c443 730c55e9 2981fce1
f172d988 fa7efc2f)

u2 31651145456365940572324876266896863292553972679075081558228163243443159960919181474327830605875067558143447293026147875690449308871770 8

(=0x8c3b4bf7 5ff5eaf5 4df2119b a413785d 73059f0b aa677e16
e6eb7cf6 1e066961 e54e4b52 6ae528b1 d3c8cf8e aa3a7df0
3b7a9a0d bb827a6f),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.2 (with the default square root function).

O.2. Example with Ed448

Pe=(x, y), k*Pe=(x1, y1), and (k+1)*Pe=(x2, y2) with Ed448:

x 127112341071454423946496045432979478879062446966923725519638164189306625397984447836475330424079449836817454081067422078812078265662747

(=0x2cc52fd1 6370554f 00c0f73f 64bda240 f5950177 d9033f6d
74acd12d 68c79a51 315f556f 240973f9 e5f71ed7 9314ee9d c87f0b1b
bcc0fd1b).

y 69251010954633529003803699627438795111055087299023774963200632446226776187009645999637901493150204695178697037386193806607746871598523 8

(=0xf3e8bb95 c9675fd0 0c388fc5 e96cfbc7 3c19d945 76849979
34c4ab60 73c4a763 c2a89bac d3879838 f4de11a3 3a4710c2 396dea1d
cc012956).

x1 692687944390887339268839580902569422568573497969223323638888137509
71700910417786272464007666020220956482611896297610096130434552586
39205

(=0xf3f8c472 ca2e730b 05cc9092 f9d40956 029113e3 e92c2d55
76406db2 c2903721 62f43371 1c0ec80c f8d7222d 1d701467 9da18531
0fb5bb65).

y1 505167074182035311590012232936232882968032995989684909150661543627854182073933232952513836331211983807543848738416143596310710340909734

(=0xb1eccbfc 5f5f92e8 d9129d14 b721c524 96fc1b1f a4c17c5f
e4979b0c 763f34ba 91299376 d2499220 19b05f56 c3bb6b5d ac988271
287d7aa6).

x2 672872621244442312431082224988499103624555909909353263630621271660412694789405598127099781962898237441602260767292345135618293810587868

(=0xecfe1a4f a4cd7e2f 19afcf16 1ce2198f 0a850beb 41afa209
94741609 5b1a858a 8e9548f5 011d188e d50484d3 119103f6 8bcd5ba2
a6e3e8dc).

y2 13744276256057290540518554008940700979716578667786691114397525367926845428757574070631798701543078825889882931670002491601144816593034341

(=0x3068a338 4016ebfd a229ac73 b5c30bba ff67e183 71d1185f
19dfbbee 28478baf 9034ebad 51407f01 35162743 c2c234bc 2d484c13
552ea565).

The representation of k and the compressed representations of Pe and k*Pe in tight LSB/lsb-order are given by

repr(k) =0x01558a4a a6647f5c 2178228a 7ba16d94 6118939b 4ac65043
d4088e88 bdecba3e f9987832 219d4e10 e3c48409 b4a0467f
535eb427 9dddcd3b;

repr(Pe) =0x6a948033 b857b69c 4308e25c c5887b2f 1c19e1cb 35d91543
c6e523ce 06d5232c 9e99216e a29b983c e3df3697 a3f11c30
0bfae693 a9dd17cf 01;

repr(k*Pe) =0x655ebe14 8e411935 bad6ddc3 6afa0d98 0449924b 6ec99489
5d2cfc6e 30d9e927 fa3e8325 f8d83f69 24a384ed 28b9489b
1749fafa 3fd3378d 01,

where the rightmost bit of the rightmost octet indicates the parity of the x-coordinate of the point of Ed448 in question (which, in this case, are both one, since x and x1 are odd). See Appendix H.3 and Appendix I for further detail on (squeezed) point compression.

The scalar representation and (squeezed) point representation illustrated above are fully consistent with the representations specified in [RFC8032]. Note that, contrary to [RFC7748], [RFC8032] requires unique representations of all elements of GF(p).

A randomized representation (t1, t2) of the point k*Pe in tight LSB/lsb order is given by

t1 39735704775900345938010207153209108583412552056119766898974760057713788148597034680608003819433647348370910486519180632600669150423154 7

(=0xde295d0e 5efceb9b f43967ca be45a54b a1f75bdd a4b1b1b3 b24a8d1d f2056329 e506867e c968aa8b 866017e4 f0cbc343 2cf8e7fa 0b202fd1)

t2 71180030153060033079106806246760018366358934059388495080813609138905625199789399589430966082776343407189730628032015104406312029606480 9

(=0x94ecb72a 069a5322 e62d9357 c49d5664 1c351611 d1f361a8 cbb8a12c f410e821 4fbe8e02 8d85d404 399b4c7c 5a6a72ce deef7b08 96302d5f),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.3 with the default square root function and underlying isomorphic mapping between Ed448 and Curve448 of Appendix M.2.

This representation can also be expressed in tight LSB/lsb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(1,1) and where

u1 79943008055528554246658339211488678620237425908117917888799033890200532749642820843532129578709445455491179906662585567756287085693163

(=0xd713005d bece883b de9e7077 e0084c74 e3f8ccf3 dcdf9af2 2db99b77 5a9c3de7 c8d14433 634cee63 531d3d85 0637c24d a28691a3 ac041438)

u2 27372897260471126095966214991707176858637173354855385604862832584772303045967066152922489073070151943109920536763943700636849997284292 5

(=0xb5a46d1d be03f21b a4070e3c 51e42a50 1de9a4e6 3155b58c 41dbdaed d5089539 cf69bbc8 78f3809d 5630ab65 c250e49b 3a91a31d 067f1606),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.3 (with the default square root function).

O.3. Example with Wei448

Pw=(X, Y), k*Pw=(X1, Y1), and (k+1)*Pw=(X2, Y2) with Wei448:

X 29070637261778856087396075817199998758219070555984737667402173284553898710776541937547992537257732413157832954298996528801181182049134 4

(=0x6663c64e 5b9a1f6d cbee3f5f 839b7dd8 6f53cc3e 0a01dab3 e4a8314e 8d54be2a 4d63380b 692381bb ed7339fd dca7b0cd a80166fe 18c15250).

Y 30578727850066757341435137807347775064915058999485530015946871157867946314072749368706185807141079316617309993502226448947292856049714 9

(=0x6bb38e82 8d52337f 6f0395ef dc16c776 52162f5e 309112ae fc7401bf 0cfb0499 eb1ed555 bf507ebc c33b4753 2d6dc6c5 d68dea1c c1e4c1fd).

X1 40351504322781497250899987383866753965468971276834772118588405333771408679393559807885734368933573692014029289580426172248960920794614 2

(=0x8e1f426a 4a1af133 ff970fe2 76693c7a eaa78786 361b1cfe 4ccbd786 e020ba9a 0bf65a1d 5d9a128a f85c63a2 79a00139 7aca56db 15341b9e).

Y1 55735504615964066386264989698774850924544182484936624265048483231356938593626278801845862824392346027980235940546117374126675437581154 7

(=0xc44e5e0f 2c254d23 1dc082db 77175e8c fd37793c 22ebe200 77905a5f 750b3c9f 4a95d4d5 4e1a1e54 d2d31689 4249252d 0c8b1c45 1c1481db).

X2 5172447138615241468712230076302665088274020590997087683492074610121508416303604179834986180511406702029983417676758930643822754281 77944

(=0xb62dc975 470cc05b 082dae0b eabe1dda 25487b2a 9663eec8 f3bd3d0e eb724f44 dc53671c ffabb9ee 0c02aa74 b083cd82 a821a4cf 6f6e5818).

Y2 5768210322358523391850734495006295030677029621527132061293720493877499282103483092990510136901415757273082719665657484294344333591 20741

(=0xcb2988a4 6e37f9a9 a7a1255b 2fd2eea9 82308e7c eb8e18b8 2252175f fd416a10 5984c6b8 36470e48 31879293 8f6139c6 f96164cb 14010965).

The representation of k and the compressed representations of Pw and k*Pw in tight MSB/msb-order are given by

repr(k) =0xdcb3bbb9 e42d7aca fe62052d 902123c7 0872b984 4c1e199f 7c5d37bd 1171102b c20a6352 d9c91886 29b685de 51441e84 3afe2665 5251aa80;

repr(Pw) =0x80 6663c64e 5b9a1f6d cbee3f5f 839b7dd8 6f53cc3e 0a01dab3 e4a8314e 8d54be2a 4d63380b 692381bb ed7339fd dca7b0cd a80166fe 18c15250;

repr(k*Pw) =0x80 8e1f426a 4a1af133 ff970fe2 76693c7a eaa78786 361b1cfe 4ccbd786 e020ba9a 0bf65a1d 5d9a128a f85c63a2 79a00139 7aca56db 15341b9e,

where the leftmost bit of the leftmost octet indicates the parity of the Y-coordinate of the point of Wei448 in question (which, in this case, are both one, since Y and Y1 are odd). See Appendix H.1 and Appendix I for further detail on (squeezed) point compression.

The scalar representation is consistent with the representations specified in [SEC1]; the (squeezed) point representation illustrated above is "new". For completeness, we include a SEC1-consistent representation of the point Pw in affine format and in compressed format below.

The SEC1-compliant affine representation of the point Pw in tight MSB/msb-order is given by

aff(Pw) =0x6663c64e 5b9a1f6d cbee3f5f 839b7dd8 6f53cc3e 0a01dab3 e4a8314e 8d54be2a 4d63380b 692381bb ed7339fd dca7b0cd a80166fe 18c15250

6bb38e82 8d52337f 6f0395ef dc16c776 52162f5e 309112ae
fc7401bf 0cfb0499 eb1ed555 bf507ebc c33b4753 2d6dc6c5
d68dea1c c1e4c1fd,

whereas the SEC1-compliant compressed representation of the point Pw in tight MSB/msb-order is given by

compr(Pw) =0x03 6663c64e 5b9a1f6d cbee3f5f 839b7dd8 6f53cc3e
0a01dab3 e4a8314e 8d54be2a 4d63380b 692381bb ed7339fd
dca7b0cd a80166fe 18c15250.

The SEC1-compliant uncompressed format aff(Pw) of an affine point Pw corresponds to the right-concatenation of its X- and Y-coordinates, each in tight MSB/msb-order, prepended by the string 0x04, where the reverse procedure is uniquely defined, since elements of GF(p) have a unique fixed-size representation. The (squeezed) compressed format repr(Pw) corresponds to the SEC1-compliant compressed format by extracting the parity bit t from the leftmost bit of the leftmost octet of repr(Pw), and replacing this leftmost octet with 0x02 or 0x03, depending on whether t=0 or t=1, respectively, where the reverse procedure is uniquely defined. For further details, see [SEC1]. Note that, due to the bit-length of the prime p, the squeezed compressed format repr(Pw) and the SEC1-compliant compressed format compr(Pw) have the same size.

A randomized representation (t1, t2) of the point k*Pw in tight MSB/msb order is given by

t1 655783099225353926682910498535559663266263823350679216116
172951494291735730803127024621397533084891460609898061397
896825551162064841608

(=0xe6f93655 2765628b accfe61c 7dc6a594 e06fb243 70195ded
74d88a53 fdedc2e8 077e0eff 62fa6a80 fa26b499 1f8796f5
21f2f03b f7e92b88)

t2 357918241879339174086992006475988394618511927120788596330
507910466738735762660894972854331591097934354210992993787
402433561014235472657

(=0x7e0ffcaf 7add27bc bb723629 95fdedd0 8769f676 78d953bc
0d38f4f6 d63a59dc 00f2d55a a4db7dab 16364503 591edcb1
e095a577 43dea311),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.1 with the default square root function.

This representation can also be expressed in tight MSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(0,0) and where

u1 27611657347368404959967397114204194300254601872574450485899921013292448115666537680136522621572543754150268605539997454394995300346621026

(=0x6140460c 1860a8cb 7c8ab942 b9509a84 95b4093c 95be5c8b df46e24c 069fe28a a23e4bfc 5bc29543 ee9ff503 febb80c8 eb207253 8d7c6c62)

u2 12869259506048775987144205470412396593822308724186376817940551256934049628653984993845772753966093264246449103736929171375605159033619 3

(=0x2d53abf2 370638a2 c2d38efe 718d0189 18d15d15 f132741b 34405174 97fc0884 0c6be3a5 d9c201b9 cb0c3637 2674078e 59ac8cd7 4f9fcec1),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.1 (with the default square root function).

O.4. Example with Wei448.1

Pw1=(X, Y), k*Pw1=(X1, Y1), and (k+1)*Pw1=(X2, Y2) with Wei448.1:

X 4141450526730296282649632386280034673014818460070631703020083167813123337737005257876668389910719145841028692415431602235556184165 13314

(=0x91ddb90a 3c19f561 21de39ad a8c6bb00 579a6d2d 9ff6b810 b109bf41 6e4e6227 0fc34010 be9ec68e 5ca11111 bc99e998 cff0f6db f4225122).

Y 2167870352469309100572852722112408324088948108923173967831193902043874709051080711177237887514058399787606848450432099149433728340 08081

(=0x4c5ac727 121de1f1 be917280 829a6d4c 9f615e3a 879a7dfd 50f8bdcc 75d5856b 1d01ffaa 44e5ba0a ed0e341d 9383e15a 6cd48db8 c1e26c11).

X1 2121173492052500182725408255711214034020810974000451926455809818940985376833176210029490012696175276046779431389727351279961384020 21113

(=0x4ab5bb5f ca80119b 6280f5d1 aec51745 23ab57ab 4d617195
38f453dd 2e8d9b66 a5417d1b ed0cee3d 4d6c84ca abda1d41 b7a805dc
cbaefef9).

Y1 14152482531219571027190110620355502977165146571026919001455348108
06142769037926777863731011790633441497896003632149582109867558046
16181

(=0x31d8b337 09272016_25d2f9d6 cb0e2396 b7088c79 ffc8571f
6dc9bfe9 9e0783d4 1f684439 c02981f1 83f6696d 9c0377c9 431b8186
f503d5f5).

X2 30394319241133688143587947164786865078477223372122681434460686381
23744153597949961703624604448300529949032402198106459850911229168
46262

(=0x6b0d487d cba3633b_034f65a5 bbd1c8eb 1b6dcb1f 8d787db1
a581c08d ad23cbcd 6faa39d5 36731645 fd2fd6c0 03367bff 9093d29d
550d6ab6).

Y2 18866191129065867707969757296934620738822864945913956797432892866
18725386530370846638505587040510045280940919798896557156654042590
85719

(=0x4272da7b 7ad66918 144ae679 3811eb6b 2124b02f 42fd51f2
34e6f3ea 6285d40e 43cf726f 585b7e74 c4448acb b0c3ab89 d5a55678
c4622d97).

The representation of k and the compressed representations of Pw1 and k*Pw1 in tight MSB/msb-order are given by

repr(k) =0xdcb3bbb9 e42d7aca fe62052d 902123c7 0872b984 4c1e199f
7c5d37bd 1171102b c20a6352 d9c91886 29b685de 51441e84
3afe2665 5251aa80;

repr(Pw1) =0x80 0x91ddb90a 3c19f561 21de39ad a8c6bb00 579a6d2d
9ff6b810 b109bf41 6e4e6227 0fc34010 be9ec68e 5ca11111
bc99e998 cff0f6db f4225122;

repr(k*Pw1) =0x80 0x4ab5bb5f ca80119b 6280f5d1 aec51745 23ab57ab
4d617195 38f453dd 2e8d9b66 a5417d1b ed0cee3d 4d6c84ca
abda1d41 b7a805dc cbaefef9,

where the leftmost bit of the leftmost octet indicates the parity of the Y-coordinate of the point of Wei448.1 in question (which, in this case, are both one, since Y and Y1 are odd). See Appendix H.1 and Appendix I for further detail on (squeezed) point compression.

A randomized representation (t1, t2) of the point k*Pw1 in tight MSB/msb order is given by

t1 30349447456627081966896308120844031142238627924834637298980090674988867944305747920755446164608334333014574668756732322837891922156528

(=0x6ae4d2fc 57e63e5e bfdc44e6 5148d1bd b30b7c7b 2ca2a66a 8a2bea6c 69113c79 7a4d6d0f 3c89b06a 3883ab2c e7d73f42 24c82419 391e9bf0)

t2 63787353416158151793816810287152364078066202035738608932814442683694785861707525682829818881711794559929694003010385886611936178650609 0

(=0xe0aa61c1 213a19b4 a9fddbb3 4c1377d0 4cd1fb84 017a1719 e57b243b 31b13406 d5d77138 23c5a1b8 4fe271a5 2e53c98f 900f2900 d1e76b6a),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.1 with the default square root function.

This representation can also be expressed in tight MSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(1,0) and where

u1 25803641311930943311352784687847668468174444543611493503637245566625939639792164542389388840655381193023798564125155167238320655039783 7

(=0x5ae20fb6 5cafa07a 40421568 72419f49 dc31cbe9 766806f6 8b1dbd7f 628c8ecf 10577848 e2e87ac2 fead0f09 6726ee34 c2ed465f 5b7be38d)

u2 19396214005242932057614051945577610949117899102334764663472356420092501244418781548440623041398010029123397592988158067111655513608240 9

(=0x4450c0ba ba9ee42a 4723b3b4 dbe7613f a78a2feb ee01752f 9f8f51d6 41476eb8 041c9d87 d1b6df7b 9c6b48ad 2cdf4c20 02d22f0c fbf521e9),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.1 (with the default square root function).

O.5. Example with Wei448.-3

Pw3=(X, Y), k*Pw3=(X1, Y1), and (k+1)*Pw3=(X2, Y2) with Wei448.-3:

X
541217938657261755059020386005621907206504566785001061681732859869999531708218763586616425010404811083912084906688745035466757984489 68

(=0xbe9f5a23 51709e13 d5ad50c2 a27be8ee 1b051970 2580d5c3 c2de7f75 3010635e d89ef547 8b67dc54 16d63c5b 1cc1116f dd453515 71b39b48).

Y
149622821013045480306278353118872758337180708189653063620069344555916877338198344570925661588752645565703405112108562276363703558012661

(=0x34b2dcc4 92d6a940 e6249c14 122d0ba4 5dc040e9 3f060d8f a65fa300 eb3cc969 25188b59 2d31039c f7a8e14a 48320a32 efe9b42b 986afef5).

X1
188082959166466458252160658472661504040624706298338548401559538586309179569677374160765979482818169238179040393575013524760598264865547

(=0x069fdd7c 2ec1ecbf d3cd0e27 1e8110c6 d2e478f2 aa393928 64a5511e da0b8dc7 3834fd57 b5ef8527 361a8176 c6da44ee 63701c0c f49d7d13).

Y1
122129452440644716343264665762573139276399042739112109534877616567768416114486537351314386830804174804782840109806066776770377984685920

(=0x2b03e68e b61581c4 9f977443 3e1ddc63 976f8f1d cdb185ee 9c53328d b425973d 359bbc09 468645c4 0996a2c7 fda561be acb4d0b5 745ab760).

X2
586729764850864361020486790937164822492966228483510515685120203199787208395010848940737083273352715484372806819550763288657408669512670

(=0xcea6f66e e741e7b3 ee50acd4 bd6eacbf 821fab72 bf5fe85b 8f614af9 04aff677 15e820b9 e4bcc159 f67a97f3 2c176d2c d9b7cdeb f753f3de).

Y2
636618999921090300512191775163784713835132174724974605179365036297952284023808054331862742814924977477310800944746629268266181828041265

(=0xe0394408 ed2b4efb b6b6ac7e bc815516 fdf31a6e d32db3f9
54cd8ac1 c7ddf0cc e7507688 a70f219a 57eef863 49003560 66747ca3
00105a31).

The representation of k and the compressed representations of Pw3 and k*Pw3 in tight MSB/msb-order are given by

repr(k) =0xdcb3bbb9 e42d7aca fe62052d 902123c7 0872b984 4c1e199f
7c5d37bd 1171102b c20a6352 d9c91886 29b685de 51441e84
3afe2665 5251aa80;

repr(Pw3) =0x80 be9f5a23 51709e13 d5ad50c2 a27be8ee 1b051970
2580d5c3 c2de7f75 3010635e d89ef547 8b67dc54 16d63c5b
1cc1116f dd453515 71b39b48;

repr(k*Pw3) =0x00 069fdd7c 2ec1ecbf d3cd0e27 1e8110c6 d2e478f2
aa393928 64a5511e da0b8dc7 3834fd57 b5ef8527 361a8176
c6da44ee 63701c0c f49d7d13,

where the leftmost bit of the leftmost octet indicates the parity of the Y-coordinate of the point of Wei448.-3 in question (which, in this case, are one and zero, respectively, since Y is odd and Y1 is even). See Appendix H.1 and Appendix I for further detail on (squeezed) point compression.

A randomized representation (t1, t2) of the point k*Pw3 in tight MSB/msb order is given by

t1 450833060883286904091316612794941178576639837300736625958
696097131313213727115363096930063001237631586932727905179
306828042642854311987

(=0x9ec9ba07 3fb2bb5e 9dbee995 067ce094 63601ecd 325f0930
aea79cb8 745fa71d 4caa37ee f04fab67 ab2de747 4ac0a025
830f4828 429cf833)

t2 339205723274519707955026734148022275762579914421865223818
363622725164496136165251928391223173879522521195772276587
373445978123589677750

(=0x7778c1f9 9d900633 d161d7ea a963ddad e9101d3f f4f04710
623d2a51 6ca10133 3db9ccc3 86df9271 fbb72740 77f79dd1
9aed0bfb e3bc72b6),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.1 with the default square root function.

This representation can also be expressed in tight MSB/msb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(0,1) and where

u1 589255274721777493669102139212346422449226408440608788354
266603544786997157375671957901717836941301424106139118763
92799989153446639329

(=0x14c11156 85eab1a5 f6c00d37 a3f6bd73 fe403dd1 31e337e7
15927c25 0264a8f8 d2cd661e b5138468 92a3b91d 09284398
17c2e361 96fa36e1)

u2 213991023129828413030692573508989139610229330687681826719
574082317313789459478773972345123463766002343322541837566
496527438452046182709

(=0x4b5eac5a 3632b273 012a1050 7762eba4 8df1ccad 16dd9e6f
d68e57a9 89de5a0c 1eda0951 e4f3de0e 39f5c37b 2f8f04d5
52c093d8 fb983935),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.1 (with the default square root function).

O.6. Example with Edwards448

Pe1=(x, y), k*Pe1=(x1, y1), and (k+1)*Pe1=(x2, y2) with Edwards448:

x 703203958930289616730466399854098702262494427017609560792989566882689660099942189775187780494684899785232536165966574428762071955867733

(=0xf7acf3ca b79b29c2 aa44863d 9edaeca4 8c90ad84 e460df42
7dd9ab59 1bd8a844 07cb3419 59309b33 1e22bfa1 a2d37e10 e2e42a1f
170f0855).

y 706287068548572816488632914879421660521379914413200552376443044645878793827316539146465352892969935075422424361399618773442407421198773

(=0xf8c2f181 3bceee8e 085ecd70 d1b6aa4c ea9b95bd 8f36ab44
c79e9124 1ea625b7 f9f5ec57 89cc5af2 a2eb255a b252b874 509dc0d9
685841b5).

x1 381258750416497012117057905542447137131349187494458545422729995967405898630448879525833497883880945625772149610576989488018565732840277

(=0x864880b9 e1900c68 ba4a545a 6fe2b161 62dcc3b9 fa218e4b feba9828 5cee5193 f2c989f6 c3b94eb6 2914dce7 b4818e4d 8fc8d51f 05a13355).

y1 1106065384661018275399116262742763170789842116683990772697836944453337541552746428662176632660036639406375548888849623833963458813 1154

(=0x03e54af3 7f4cf5e6 5f1e2acd 5c4a4554 76adc652 b198ab2a 719e5aa9 ee749871 0193da82 ab6d000b f55836b1 0615653f 69514297 f4459f52).

x2 1562050378841349704480451730402152443906237448982254772850833793750606335270276724725939683726318058744384611584731365019896485812 8760

(=0x05806f71 95e85352 ef3960ac 1ff9cf6c 3c99e0ee 2e75edfc a133cafc 4a4b5fbf e4339859 c5fa123b 70ad2faf 7584ab9d 264540e7 7d560978).

y2 4001991751412172746312219012568937789070357069833715815915351083668442386516751945577468473801561261386285902585868517988506010293 44096

(=0x8cf44811 3cec6e07 d1bbe9f5 4062075c 6fec0ac5 31272dce 1f446aeb d895373d e312c18d 6a345755 2861e014 0cc23158 a46ace4c 9ca21b60).

The representation of k and the compressed representations of Pe1 and k*Pe1 in tight LSB/lsb-order are given by

repr(k) =0x01558a4a a6647f5c 2178228a 7ba16d94 6118939b 4ac65043 d4088e88 bdecba3e f9987832 219d4e10 e3c48409 b4a0467f 535eb427 9dddcd3b;

repr(Pe1) =0xad821a16 9b03b90a 2e1d4a4d 5aa4d745 4f5a3391 ea37af9f eda46578 248979e3 22d56cf1 bda9d957 32556d8b 0eb37a10 717773dc 818f431f 01;

repr(k*Pe1) =0x4af9a22f e9428a96 fca6a860 8d6c1aaf d000b6d5 415bc980 8e192e77 955a798e 54d5198d 4a63b56e 2aa2523a b35478fa 67af32fe cf52a7c0 01,

where the rightmost bit of the rightmost octet indicates the parity of the x-coordinate of the point of Edwards448 in question (which, in this case, are both one, since x and x1 are odd). See Appendix H.3 and Appendix I for further detail on (squeezed) point compression.

The scalar representation and (squeezed) point representation illustrated above are fully consistent with the representations specified in [RFC8032]. Note that, contrary to [RFC7748], [RFC8032] requires unique representations of all elements of GF(p).

A randomized representation (t1, t2) of the point k*Pe1 in tight LSB/lsb order is given by

t1 12539004885888740010407478787940283385185473933983609373373463877675598302103421205841589128835026570110121998169884908961281385104200407

(=0xed921f3d 6ea4e452 dd06e783 782cbeb3 c5847a79 d9e6b993 bd387cf5 feeddafe af8c038d f2732362 92724d37 273eedfc f2ab2499 98a79434)

t2 49332485847848124240501842386555063850771545465413551416884256014982736076338288919996398005697989591854528088324778700399779828693147310

(=0xd53a5125 193b6ab9 8db48161 20fb4865 02cf0546 3b48d8a6 514af28f 43c026cb 0f2ff3d5 e558bb03 4b833cd1 1ca710cc 9bf0c2a3 351083b5),

where this representation is defined in Appendix K.5 and uses the mapping of Appendix K.3.3 with the default square root function and underlying 4-isogenous mapping between Edwards448 and Curve448 of Appendix N.2.

This representation can also be expressed in tight LSB/lsb order as the pair ((u1,s1),(u2,s2)), where (s1,s2):=(0,0) and where

u1 13599358230805971087111806770565183158421599241551117472725553364103381631905298947727648798199895770638239125450448451070842833065141388

(=0x31276f5f d399d1cd 5d18c46a eba5388f 93effaf7 9574b23b ce34ba45 5050c160 477ae803 9c3112be 596281a7 b7ae4da6 e9dd7688 191fa7f4)

u2 30072593637984721592900227552563322957603470767162046314362639383266043602775973709763778675309588088519936868686318778944917973042647

(=0xb65c5ee8 597b5b55 a87e266f b9c1f5cb 5d224ec3 8fb22f32 b0378e70 47ecc389 9585b06e 7fb4f70b 38a3b453 ab5c03d8 37b5093b 9a4cd796),

where this uses the default completed mapping defined in Appendix K.6 and the mapping of Appendix K.4.3 (with the default square root function) and underlying 4-isogenous mapping between Edwards448 and Curve448 of Appendix N.2.

Appendix P. Random Integers in Z_n

Any probability distribution on the interval [0,N-1] can be converted to a probability distribution on [0,n-1], via a suitable function that maps inputs from the source distribution [0,N-1] to values in the interval [0,n-1]. We consider three such functions, each with the property that if the source distribution on [0,N-1] is statistically close to the uniform distribution, then so is the output distribution on [0,n-1]. (Here, we assume n and N to be integers of cryptographic interest, so large.) In practical applications, one can use these functions to convert the output of a cryptographically strong random bit generator (where N is a power of two and after conversion of the random bit string to an integer via the BS2I mapping of Appendix I.2) to a pseudo-random integer in the interval [0,n-1], where the bias is small if N is suitably picked.

We consider mappings that convert an output of the source distribution to an integer in the interval [0,n-1] via modular reduction (Appendix P.1), via scaling (Appendix P.2), or via a membership test (Appendix P.3). For suitably picked N values and not too poor source distributions, the first two mappings never fail and any bias introduced by the conversion process can be made negligible in practice, while the third mapping (if it does not fail) inflates the bias by a small factor only in practice. (For details, see the remarks following each of the mappings below.)

NOTE: Each of the mappings below may yield a zero output value. One can modify each such mapping to always yield nonzero outputs, by setting output x to 1 if the original mapping would yield x=0 for a specific input y and leaving the mapping the same otherwise (henceforth called the modified conversion function). This modification has negligible impact on the bias and does yield a conversion function to integers in the interval [1,n-1]. A similar remark applies if n=h*n1, where h is a small integer: in that case, one can locally modify each mapping to always yield outputs in the interval [0, h*n1-1] that are not divisible by n1, simply by setting output x to x+1 if the original mapping would otherwise yield x=0 (mod n1). (Notice that both modifications coincide if h=1.) These modifications may be useful if one wishes to generate integers in an interval of size n and where one wishes to avoid specific output values (e.g., if one wishes to generate high-order points of a curve of order h*n1, with co-factor h (see Appendix B.1)). For simplicity,

we again refer to this as "the" modified conversion function (or h-modified conversion function, if h is not clear from context).

P.1. Conversion to Integers in Z_n via Modular Reduction

This function maps each integer y in the interval [0,N-1] to its remainder modulo n, i.e., y is mapped to x:= y (mod n).

One can show that the bias introduced by this conversion function is at most epsilon:=2*rho*(1-rho)/(N/n), where r:=N (mod n) and where rho:=r/n. Details are out of scope.

Note that if n does not divide N, this invariably introduces some bias, no matter the quality of the source distribution. In particular, the statistical distance of the distribution on Z_n can be much larger than the statistical distance of the source distribution on Z_N, since the bias introduced by the modular reduction step may be significantly larger than the bias of the source distribution on Z_N if the value rho above is not close to zero or one and if n/N is not sufficiently small. The maximum bias is, however, easy to determine from n and N. In particular, if the bit-length of N is sufficiently larger than that of n, the bias introduced by the modular reduction operation is negligible in practice. The same holds if N is close to a multiple of n (e.g., if n is close to a power of two and the input distribution is generated by a high-quality random bit generator with outputs of fixed bit-length).

Note: In practice, one does not determine the maximum bias epsilon from n and N, but rather specifies a required upper bound (usually set to a value at most 2^{-64}) for epsilon and subsequently determines the minimal value of N (where N is a power of two) for which this upper bound indeed applies, as a function of n. Table 4 illustrates this for several curves of practical interest.

Curve	eps0=2^{-64}	eps0=2^{-100}
NIST P-224 [FIPS-186-4]	224	224
NIST P-256 [FIPS-186-4]	288	352
NIST P-384 [FIPS-186-4]	384	384
NIST P-521 [FIPS-186-4]	521	521
brainpoolP224r1 [RFC5639]	287	323
brainpoolP256r1 [RFC5639]	319	354
brainpoolP320r1 [RFC5639]	379	417
brainpoolP384r1 [RFC5639]	445	482
brainpoolP512r1 [RFC5639]	575	608
Curve25519 [RFC7748]	252	252
Wei25519 [Appendix E.3]	252	252
Wei25519.2 [Appendix G.3]	252	252
Wei25519.-3 [Appendix G.3]	252	252
Curve448 [RFC7748]	446	446
Wei448 [Appendix M.3]	446	446
Wei448.1 [Appendix N.3]	446	446
Wei448.-3 [Appendix N.3]	446	446
secp256k1.m [Appendix L.3]	256	256

Table 4: Minimum value of m for which the bias (epsilon) introduced by converting integers in Z_N, where N:=2^m, to integers in Z_n via modular reduction or via scaling is lower than the indicated eps0 value, for some curves of practical interest (where n is the order of the base point of the curve in question).

P.2. Conversion to Integers in Z_n via Scaling

This function maps each integer y in the interval [0,N-1] to the integer x:=floor(n*y/N), where the floor function rounds real numbers downwards to an integer (i.e., floor(z) is the unique integer i for which z is an element of the interval [i,i+1) of real numbers).

One can show that the bias introduced by this conversion function is at most epsilon:=2*rho*(1-rho)/(N/n), where r:=N (mod n) and where rho:=r/n. Details are out of scope.

The same remarks as in Appendix P.1 apply.

Note: this mapping corresponds to interpolation on the line with endpoints (0,0) and (N,n), where values are truncated to integers. The division operation in this conversion function reduces to a binary string truncation operation if N is a power of two (which is often the case in practice). See also [comm-FIPS-186-5], pp. 80-82.

P.3. Conversion to Integers in Z_n via the Discard Method

This function (defined for N at least n) is the identity map on the interval [0,n-1] and fails for each integer y outside this interval.

One can show that the statistical distance of the distribution on Z_n is at most roughly N/n times as large as the statistical distance of the source distribution on Z_N (if the latter is relatively negligible compared to n/N). Details are out of scope.

Note that, under the above conditions, if N:=2^m and if n has bit-length m, this conversion function fails with probability 1- n/N (which is at most 1/2) and, if it succeeds, does not inflate the statistical distance by more than (roughly) a factor two.

Appendix Q. ECDSA signatures

The ECDSA signature scheme is specified in FIPS Pub 186-4 [FIPS-186-4], ANSI X9.62-2005 [ANSI-X9.62], SEC 1 [SEC1], and many other standards and can be instantiated with suitable combinations of short-Weierstrass curves and hash functions (that satisfy particular cryptographic criteria). Despite its wide-spread use, some details seem less well-understood. We, therefore, provide a concise specification of ECDSA (for short-Weierstrass curves defined over a prime field GF(p)) and give some examples of ECDSA computations where the underlying short-Weierstrass curve has co-factor h>1 and where the bit-length of the domain parameter n differs from the digest size of the used hash function, illustrated with the curves Wei25519 and Wei448 introduced in this document. Our description is consistent with all forementioned standards.

The signing operation takes as inputs a message m (represented as a bit string) and a private key d in the interval [1,n-1] and produces as output a signature, which is an ordered pair (r, s) of integers in the interval [1,n-1], where n is the order of the base point G of the curve in question. The signature verification operation takes as inputs a message m, a public key Q, and a signature (r,s) and produces as output the value "valid" or "invalid", depending upon whether the message was purportedly signed by a holder of the private key of the public-private key pair (d, Q) for the curve used with the signature scheme. Full details are provided below, where we denote the applicable hash function by H.

Q.1. ECDSA Signing Operation

The signing operation involves the following steps:

a. Generate a random ephemeral public-private key pair (k, R:=k*G), by generating a random integer k in the interval [1,n-1] and computing R:=k*G (see, e.g., Appendix B.1);

b. Compute k1:=(1/k) (mod n) (see, e.g., NOTE 1 of Appendix K.2);

c. Set xR to the x-coordinate of the (affine) point R, convert this element of the field GF(p) to the integer r0 in the interval [0,p-1], and set r:= r0 (mod n), where xR is converted to r0 by subsequently using the FE2OS and OS2I mappings of Appendix I.5 and Appendix I.3, respectively;

d. Compute the hash value E:=H(m) according to the applicable hash function H, where E is a bit string of length hashlen (the digest size of H);

e. Represent E as the integer e in the interval [0, 2^l-1], where e is the integer representation of the l-prefix of E, using the BS2I mapping of Appendix I.2, and where l is the bit-length of n. (For a definition of the l-prefix, see Appendix I.1);

f. Compute s:= k1*(e+ r*d) (mod n). Securely destroy k and k1;

g. Return to the first step (Step a) if r and s are not both integers in the interval [1,n-1];

h. Output the ordered pair (r, s) as the signature.

Q.2. ECDSA Verification Operation

The verification operation involves the following steps:

a. Check that the purported signer's public key Q is a point of the curve in question of order n (and output "reject" if this is not the case);

b. Check that the coordinates of the purported signature (r, s) are both integers in the interval [1,n-1] (and output "reject" if this is not the case);

c. Compute the hash value E:=H(m) according to the applicable hash function H, where E is a bit string of length hashlen (the digest size of H);

d. Represent E as the integer e in the interval [0, 2^l-1], where e is the integer representation of the l-prefix of E, using the BS2I mapping of Appendix I.2, and where l is the bit-length of n. (For a definition of the l-prefix, see Appendix I.1);

e. Compute s1:=(1/s) (mod n) (see, e.g., Appendix K.2); compute u:= e*s1 (mod n) and v:= r*s1 (mod n);

f. Compute the point R':= u*G+v*Q. Check whether R' is the identity element O of the curve (and output "reject" if this is the case);

g. Set xR' to the x-coordinate of the (affine) point R', convert this element of the field GF(p) to the integer r0' in the interval [0,p-1], and set r':= r0' (mod n), where xR' is converted to r0' by subsequently using the FE2OS and OS2I mappings of Appendix I.5 and Appendix I.3, respectively;

h. Output "accept" if r'=r; output "reject" otherwise.

NOTE 1: For prime-order curves, r generally uniquely represents the x-coordinate of R (since, by the Hasse bound, |E|=n is relatively close to p). For curves with co-factor h>1, this result holds only if one would know r0 (mod h*n), rather than r:= r0 (mod n).

NOTE 2: If an ECDSA signature (r, s) is valid for a particular message m and public key Q, then so is (r,-s) -- the so-called malleability. Note that this corresponds to changing the ephemeral signing key pair (k, R) in the first step of the signing operation to (-k, -R), where the y-coordinates of R:=(xR, yR) and -R=(xR, -yR) have different parity (see Appendix H). Since any party (not just the signer) can recompute the ephemeral signing key R' from a valid signature, since R':=(1/s)(e*G+r*Q), this implies that any party can retroactively put the ECDSA signature in a form where the y-coordinate of the ephemeral signing key R has a fixed parity. This observation can be used to put ECDSA signatures in a form that generally allows unique and efficient recovery of R from r for prime-order curves (due to NOTE 1 above) and more efficient signature verification methods. Further details are out of scope.

Q.3. Representation Examples ECDSA

We present some examples of ECDSA computations, when used with curve Wei25519 and SHA-256 (see Appendix Q.3.1), with Wei25519 and SHAKE128 with output size d0=256 (see Appendix Q.3.2), and with Wei448 and SHAKE256 with output size d0=512 (see Appendix Q.3.3). In each case, we indicate the signer's public key Q:=d*G, the ephemeral signing key R:=k*G ,the message m that is signed, and some intermediate values in the ECDSA signing operation resulting in signature (r,s). We write R:=(xR, yR) and Q:=(xQ, yQ), and include the ASCII string corresponding to message m. Note that the domain parameter n of curve Wei25519 has bit-length l:=253, whereas the corresponding domain parameter for Wei448 has bit-length l:=446 (which, in either case, differs from the digest size of the underlying hash function).

For completeness, we also provide an example of ECDSA computations, when used with NIST curve P-256 (see [FIPS-186-4]) and hash function SHA-256 (see [FIPS-180-4]), where the domain parameter n of the curve in question has bit-length l:=256, which is equal to the digest size of the underlying hash function (see Appendix Q.3.4). For details of the encoding of ASCII strings as bit strings, see Appendix I.4.

Q.3.1. Example of ECDSA with Wei25519 and SHA-256

d 4794127466002913886439634794756890877495119501721228452477708046179444885588

(=0x0a996146 d73d096f 6a606ad8 72e11b12 ce973033 524591c3 ebcc630d b6368854).

xQ 3442255739368936964809531240580393343360656847619747755429333773387341283644

(=0x079c3f69 9b688181 69038c35 39c11eb5 96d09f5b 12a242b4 ce660f13 3368c13c).

yQ 7698166198291735163093751722241272913088236885813432215648576219567913357634

(=0x110501f6 1dff511e d6c4e9b9 bfd5acbe 8bf043b8 c3e381dd f5771306 479ad142).

k 1742654760287647058719177782531702769875263627927591937555936092953735113209

(=0x03da4ec1 8dc83b53 5ab8857c bbd289ae 40e6d25b ba52923c e6b217a0 348ca9f9).

xR 3823654488094609767579863803266918618950131993094679963518622625371011711679

(=0x54891e12 88cf078e f3f1444c c1919e30 67eb5dd6 1c6f45d1 94b9c0e1 192d7caf).

yR 2412017513925612125626715843778697519758714347557021298122166479161455161196

(=0x3553890b d265d561 032e2daa 10b9820c 4845dbf8 f6b4f432 08f5df99 c375da40).

r 2051516994284786605932705217454214985215738134047261605176471562282845886734

(=0x04891e12 88cf078e f3f1444c c1919e2f ff907c7c ed9935a1
dc5dd15d 4860590e).

1/k 41122695303709273156068243481808769134600808188172269288861174824
34446546266

(=0x0917764a 5a76024b e9608472 bfec99be 0cffacbe 0a5a6805
0e4e75bc 36a0d55a).

m "example ECDSA w/ Wei25519 and SHA-256"

(=0x65 78616d70 6c652045 43445341 20772f20 57656932 35353139
20616e64 20534841 2d323536).

E 10340924651306471157182528854495725311608440786255119926874295925
4624066081637

(=0xe49f8f34 0ac7fd87 1ca6c035 1ac83b97 2ec4711e f4a79d37
214b6b94 c6f41365).

e 12926155814133088946478161068119656639510550982818899908592869906
828008260204

(=0x1c93f1e6 8158ffb0 e394d806 a3590772 e5d88e23 de94f3a6
e4296d72 98de826c).

s 18145968192643101430203980459406244543409512911444833316246990876
74236833451

(=0x04030680 d490837e 0b50800d 5052feb3 8181da43 f14fea65
d75fff8e 095d8eab).

The ECDSA signature (r,s) can be represented uniquely as the 64-octet string

0x04891e12 88cf078e f3f1444c c1919e2f ff907c7c ed9935a1 dc5dd15d
4860590e

04030680 d490837e 0b50800d 5052feb3 8181da43 f14fea65 d75fff8e
095d8eab,

where this string is the right-concatenation of the integers r and s, each represented as fixed-size octet string in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6. Since an ECDSA signature (r, s) is valid only if the ECDSA signature (r,-s) is, one can alternatively use the representation

0x04891e12 88cf078e f3f1444c c1919e2f ff907c7c ed9935a1 dc5dd15d
4860590e

0bfcf97f 2b6f7c81 f4af7ff2 afad014c 935d1f9a b1a7b270 80b2638c
53984542,

with the same representation conventions.

Q.3.2. Example of ECDSA with Wei25519 and SHAKE128

d 500321305808554198700692685210796365340511056940263150735113747092312944544

(=0x0b0fb7de 7b857528 c16cc691 f91acb6a e6f83700 c2257210
d9ce4a66 540f5c44).

xQ 4967487257561811564960530186009752473969138625538798968928441210571525081583 6

(=0x6dd2fb44 ebc47199 0558875c 338b32a0 01c04e5e 54b0239f
931ba404 43fee35c).

yQ 1966875207901497624624966250672264423130801987201384593610136465688265305151 4

(=0x2b7c1e81 e0d7311a 7e73c581 ac8d7478 f5d8402e a25ecf03
2fcf49b3 ebe3ba7a).

k 6745822859353803986803117518353782335342787778315854615124514020451058711301

(=0x0eea001c 69e39d65 a93a736f 51dab17d 3c89d712 67b95dba
28f43e6c 6d73fb05).

xR 2271079352831674441450281971268228387695642357612612226298464500765688945778 7

(=0x3235da86 6c184868 db1060f4 c57414ba f9dd8bbf af94eb8e
65a26fa8 146d9c7b).

yR 4822838611594794238011785034040651407700833383638071570166321997159492095419 6

(=0x6aa04c98 30a51d5a 226fc67b 6ec00aa4 66eae465 432825e3
c8da192d 330c8954).

r 9997767963199577725832600235533011543850744979863994449787921928005267048207

(=0x0235da86 6c184868 db1060f4 c57414ba bb409e23 c6ae150b
5d6b4658 fd8c20b4).

1/k 16237902548817115200666748510759761693156732885271500846541777492
82633956147

(=0x03970860 022244d0 1cee5f2e 973372d7 2000b51d 2d75731c
0e27428a 7e723b33).

m "example ECDSA w/ Wei25519 and SHAKE128"

(=0x6578 616d706c 65204543 44534120 772f2057 65693235 35313920
616e6420 5348414b 45313238).

E 375584811861750986062789709110210569164720383200898751225035375 02
754552388537

(=0x530958d6 432ba571 6a20fd9b d3592234 943da1d6 57f55f07
2ab01860 3f9f9bb9).

e 469481014827188732578487136387763211455900479001123439031294218 78
44319048567

(=0x0a612b1a c86574ae 2d441fb3 7a6b2446 9287b43a cafeabe0
e556030c 07f3f377).

s 609790349740355062606454620982554018778989287301774158444893762 61
15704732698

(=0x0d7b4a83 96b37670 d6ef4ac7 6ce69d43 a65859de 4ecbe649
f56a7a1f 7fb5bc1a).

The ECDSA signature (r,s) can be represented uniquely as the 64-octet string

0x0235da86 6c184868 db1060f4 c57414ba bb409e23 c6ae150b 5d6b4658
fd8c20b4

0d7b4a83 96b37670 d6ef4ac7 6ce69d43 a65859de 4ecbe649 f56a7a1f
7fb5bc1a,

where this string is the right-concatenation of the integers r and s, each represented as fixed-size octet string in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6. Since an ECDSA signature (r, s) is valid only if the ECDSA signature (r,-s) is, one can alternatively use the representation

0x0235da86 6c184868 db1060f4 c57414ba bb409e23 c6ae150b 5d6b4658
fd8c20b4

0284b57c 694c898f 2910b538 931962bc 6e86a000 542bb68c 62a7e8fa
dd4017d3,

with the same representation conventions.

Q.3.3. Example of ECDSA with Wei448 and SHAKE256

d 837739218338830657241527550407799263247010426676801377623292411159259716037644412069924186291014195586621722463056076559589057222796690

(=0x1d818b12 92af6ef4 3f0ed657 b55d2ab7 a0cd1e64 516414d1 d32ea610 dd6dddbe af65bc96 df648e6d fac1b907 6588b37e 984d5860 7390970a).

xQ 403515043227814972508999873838667539654689712768347721185884053337714086793935598078857343689335736920140292895804261722489609207946142

(=0x8e1f426a 4a1af133 ff970fe2 76693c7a eaa78786 361b1cfe 4ccbd786 e020ba9a 0bf65a1d 5d9a128a f85c63a2 79a00139 7aca56db 15341b9e).

yQ 557355046159640663862649896987748509245441824849366242650484832313569385936262788018458628243923460279802359405461173741266754375811547

(=0xc44e5e0f 2c254d23 1dc082db 77175e8c fd37793c 22ebe200 77905a5f 750b3c9f 4a95d4d5 4e1a1e54 d2d31689 4249252d 0c8b1c45 1c1481db).

k 564630342353061690148823075620361130959668449176312986867495715742289590975693311561472435157514419088439772050424912144493814086534243424

(=0x13e308f8 2f7eb169 78a86240 a2087c59 38ad954c 5a725311 00e2738b 93f87064 06846d1b 0348c213 5cd8f9db 21cbf970 6b70fa40 29364070).

xR 464211175292234359405903992000910232588801553953469293422284755778741191715457269486889118734630064318765372865405250982715920129560118

(=0xa37ff9a3 9734a3dc 9ccf72b7 cb3b8e5e 20d4e1f1 655c973a
c72e4aa4 6f139529 84b1cd37 2524bf09 c4e38684 5c88cc79 e8e19242
42398e36).

yR 48450878695819342796480063527087959345962966106444727188216313803
37436540801561730584163096514114057681225129685101546366763700225
61560

(=0xaaa6202c df8711b2 6e5a8802 6c5d86b3 2f320d89 8f48a809
40818982 bb74e0cc 7b884f20 aad090fb 90c4c93f fd84ed56 c03451d8
84fc7718).

r 10079181314443091413124208805690796541198087360981026328153965618
84491837923780980544976081512453123921447854472219263731684084102
60560

(=0x237ff9a3 9734a3dc 9ccf72b7 cb3b8e5e 20d4e1f1 655c973a
c72e4aa5 757f4d55 fc1416a3 c77851e9 820a019f 40fdadcf a1f00d1c
eb890450).

1/k 13511362508598651506450197334516130806445911047753884276726477993
82054003440714897722657048821186399503939251111689038388764827779
24830

(=0x2f96a107 4a355722 1f20fd90 aed12db3 83b3c32f 593079f4
779e2942 3ad2b5e6 0ea15bdc a57e5827 04ed1f09 e42b8352 68428208
502444de).

m "example ECDSA w/ Wei448 and SHAKE256"

(=0x6578616d 706c6520 45434453 4120772f 20576569 34343820
616e6420 5348414b 45323536).

E 70518111636481318756745253634149479528712660170653218748970032198
09200302878838207999610865248047293428251821985248449616335015355
07446207035858319677416 5

(=0x86a488f3 62da4be6 147ef640 34ad43ca 3fde6613 456cc034
55555f34 c34778b9 02f05145 62e2113c f4894daf 4446bb10 636b43df
fa5e2434 b1262bee 420fef15).

e 95569862294810470138539721873942452276898897320233558414699702549
60123138804500061287999759232209252819044656295606728178066107449
5757

(=0x21a9223c d8b692f9 851fbd90 0d2b50f2 8ff79984 d15b300d
155557cd 30d1de2e 40bc1451 58b8844f 3d22536b d111aec4 18dad0f7
fe97890d).

s 152568391627380571004635203321299587986731062444667642768825164376531673186103711939456185869660919303123029096598122890652967253692957

(=0x35bc73be 7e3fd7a1 de9fdb4f be96cbcd 0bb9beec f8286d04 026f3440 d6e68e25 9916ea2b c4407e9a 83ecf91a 8473c1a1 cda742d0 dab5d21d).

The ECDSA signature (r,s) can be represented uniquely as the 112-octet string

0x237ff9a3 9734a3dc 9ccf72b7 cb3b8e5e 20d4e1f1 655c973a c72e4aa5 757f4d55 fc1416a3 c77851e9 820a019f 40fdadcf a1f00d1c eb890450

35bc73be 7e3fd7a1 de9fdb4f be96cbcd 0bb9beec f8286d04 026f3440 d6e68e25 9916ea2b c4407e9a 83ecf91a 8473c1a1 cda742d0 dab5d21d,

where this string is the right-concatenation of the integers r and s, each represented as fixed-size octet string in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6. Since an ECDSA signature (r, s) is valid only if the ECDSA signature (r,-s) is, one can alternatively use the representation

0x237ff9a3 9734a3dc 9ccf72b7 cb3b8e5e 20d4e1f1 655c973a c72e4aa5 757f4d55 fc1416a3 c77851e9 820a019f 40fdadcf a1f00d1c eb890450

0a438c41 81c0285e 216024b0 41693432 f4464113 07d792fb fd90cbbe a5e395c4 2b37f11d ea95b7f5 9d7fc958 0951cdb3 55d17fc1 d0a272d6,

with the same representation conventions.

Q.3.4. Example of ECDSA with P-256 and SHA-256

d 64502400493437371358766275827725703314178640739253280897215993954599262549170

(=0x8e9b109e 719098bf 980487df 1f5d77e9 cb29606e bed2263b 5f57c213 df84f4b2).

xQ 57807358241436249728379122087876380298924820027722995515715270765240753673285

(=0x7fcdce27 70f6c45d 4183cbee 6fdb4b7b 58073335 7be9ef13 bacf6e3c 7bd15445).

yQ 90436541859143682268950424386863654389577770182238183823381687388274600502701

(=0xc7f144cd 1bbd9b7e 872cdfed b9eeb9f4 b3695d6e a90b24ad
8a462328 8588e5ad).

k 109173169016148564590757434615034534019563159944366276147140694722195121475236

(=0xf15dd2ec 0c1f92a1 0b60543c 20ccc85a 6bc502fc c8d1fa0f
cc4e0efd 7e5b8ea4).

xR 67018809247931167566425050558236675606398890455759429072257638117889035894 77

(=0x0ed12153 79636c48 3c2f7f15 5807d402 a3b22803 3af97c7e
17819ac3 169ea665).

yR 836899925592229764353479845397876327193859229649183012286088601258421928693 97

(=0xb906db6f 843c3944 597cf51b b804f66b f66690df 75f9e046
d5d4f548 870a2815).

r 67018809247931167566425050558236675606398890455759429072257638117889035894 77

(=0x0ed12153 79636c48 3c2f7f15 5807d402 a3b22803 3af97c7e
17819ac3 169ea665).

1/k 113255571212015868908722859561080225338712372995492268864584552517263129747627

(=0xfa6461b5 646b7d69 893c8e10 96376336 078815e5 9c225d70
f677307c 653ea0ab).

m "eyJhbGciOiJFUzI1NiJ9.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ"

(=0x65794a 68624763 694f694a 46557a49 314e694a 392e6579 4a706333
4d694f69 4a716232 55694c41 304b4943 4a6c6548 41694f6a 457a4d44
41344d54 6b7a4f44 41734451 6f67496d 68306448 41364c79 396c6547
46746347 786c4c6d 4e766253 39706331 39796232 3930496a 7030636e
566c6651).

E 152769562524567992932124119370895067098110394365166768305752916284259466593 74

(=0x21c67368 f436577f 447f8051 62ca13b8 0d046a3f e467247e
65ea477a a750fa2e).

e 1527695625245679929321241193708950670981103943651667683057529162842594665937 4

(=0x21c67368 f436577f 447f8051 62ca13b8 0d046a3f e467247e 65ea477a a750fa2e).

s 8912335365709302147736668478493290158013824367008962758281723900191497540936 5

(=0xc50a07d3 8c3c70e5 d8f12daf 084a5480 a66590c5 f293509a 8f3f7f8a 83a354d5).

The ECDSA signature (r,s) can be represented uniquely as the 64-octet string

0x0ed12153 79636c48 3c2f7f15 5807d402 a3b22803 3af97c7e 17819ac3 169ea665

c50a07d3 8c3c70e5 d8f12daf 084a5480 a66590c5 f293509a 8f3f7f8a 83a354d5,

where this string is the right-concatenation of the integers r and s, each represented as fixed-size octet string in tight MSB/msb-order using the ZnE2OS mapping of Appendix I.6. Since an ECDSA signature (r, s) is valid only if the ECDSA signature (r,-s) is, one can alternatively use the representation

0x0ed12153 79636c48 3c2f7f15 5807d402 a3b22803 3af97c7e 17819ac3 169ea665

3af5f82b 73c38f1b 270ed250 f7b5ab7f 168169e7 b4844dea 647a4b38 78bfd07c,

with the same representation conventions.

NOTE: The example above corresponds to the JSON Web Signature example of Appendix A.3 of [RFC7515], where the base64url encoding of the public-private key pair (d, Q:=d*G) above is given by

d "jpsQnnGQmL-YBIffH1136cspYG6-0iY7X1fCE9-E9LI";

xQ "f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU";

yQ "x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0",

where the base64url encoding of the ECDSA signature (r,s) is given by

"DtEhU3ljbEg8L38VWAfUAqOyKAM6-Xx-
F4GawxaepmXFCgfTjDxw5djxLa8ISlSApmWQxfKTUJqPP3-Kg6NU1Q",

and where the base64url encoding of the alternative ECDSA signature (r,-s) is given by

"DtEhU3ljbEg8L38VWAfUAqOyKAM6-Xx-
F4GawxaepmU69fgrc8OPGycO0lD3tat_FoFp57SETepkeks4eL_QfA".

Here, the JSON Web Signature is represented as the string m.sig, where the message field m and the signature field sig are separated by the "." symbol, where m assumes the value in our example and where the signature assumes the base64url encoding of one of the ECDSA signatures (r,s) or (r,-s), as indicated above. For precise details regarding JWS encodings, we refer to [RFC7515].

Author's Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com