lwig
Internet-Draft
Intended status: Informational
Expires: November 16, 2019

R. Struik
Struik Security Consultancy
May 15, 2019

Alternative Elliptic Curve Representations
draft-ietf-lwig-curve-representations-05

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 using existing implementations of, e.g., ECDSA and ECDH using NIST prime curves.

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 RFC 2119 [RFC2119].

Status of This Memo

Copyright Notice

Table of Contents

1. Fostering Code Reuse with New Elliptic Curves

It is well-known that 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 model. This document specifies an alternative representation of points of Curve25519, a so-called Montgomery curve, and of points of Edwards25519, a so-called twisted Edwards curve, which are both specified in [RFC7748], as points of a specific so-called "short" Weierstrass curve, called Wei25519. We also define how to efficiently switch between these different representations.

Use of Wei25519 allows easy definition of new 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).

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. 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 K.

4. Examples

4.1. Implementation of X25519

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 D.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 co-factor Diffie-Hellman key agreement (denoted by ECDH25519) results if one keeps inputs (key contributions) and outputs (shared key) in the short-Weierstrass format (and, hence, does not perform Step (3) above).

NOTE: At this point, it is unclear whether this implies that a FIPS-accredited module implementing co-factor Diffie-Hellman for, e.g., P-256 would also extend this accreditation to X25519.

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 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 hereof (which uses the v-coordinate of the base point of Curve25519 as well). For details, see Appendix C.1.

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, 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).

4.4. Other Uses

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 the resource and maintenance cost of implementations supporting algorithm agility [RFC7696] at bay.

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:

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 I for details);

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 MSB/msb conventions. Since each pair of conventions is different (see Appendix J for details and Appendix K for examples), this does necessitate bit/byte representation conversions;

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 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. This isogeny has degree l=47 and requires roughly 9kB of storage for isogeny and dual-isogeny computations (see the tables in Appendix H). Note that storage would have reduced to a single 64-byte table if only the 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 the quadratic twist of Curve25519 (i.e., 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-size (typically, close to a multiple of a machine word). Depending on the implementation strategy, the bit-size 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 (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 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 special structure of the underlying field or other specific domain parameters (since generic). Reuse of code, therefore, may result in a less computationally efficient curve implementation than would have been possible if the implementation had specially targeted Curve25519 or Edwards25519 alone. 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 7).

7. 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).

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

8. Privacy Considerations

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

9. IANA Considerations

An object identifier is requested for curve Wei25519 and its use with ECDSA and co-factor ECDH, using the representation conventions of this document.

There is *currently* no further IANA action required for this document. New object identifiers would be required in case one wishes to specify one or more of the "offspring" protocols exemplified in Section 4.4.

9.1. 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 (where OKP uses the squeezed MSB/msb representation of this specification);
Description:	short-Weierstrass curve Wei25519;
Reference:	Appendix E.3 of this specification;
Recommended:	Yes.

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

9.2. COSE Algorithms Registration (1/2)

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

Name:	ECDSA25519;
Value:	TBD (Requested value: -1);
Description:	ECDSA w/ SHA-256 and curve Wei25519;
Reference:	Section 4.3 of this specification;
Recommended:	Yes.

9.3. COSE Algorithms Registration (2/2)

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

Name: ECDH25519;

Value: TBD (Requested value: -2);

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

Reference: Section 4.1 of this specification (for key derivation, see Section 11.1 of [RFC8152]);

Recommended: Yes.

9.4. JOSE Elliptic Curves Registration

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

Curve Name: Wei25519;

Curve Description: short-Weierstrass curve Wei25519;

JOSE Implementation Requirements: optional;

Change Controller: IANA;

Reference: Appendix E.3 of this specification.

9.5. 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: ECDSA25519;

Algorithm Description: ECDSA w/ SHA-256 and curve Wei25519;

Algorithm Usage Locations: alg;

JOSE Implementation Requirements: optional;

Change Controller: IANA;

Reference: Section 4.3 of this specification;

Algorithm Analysis Documents: Section 4.3 of this specification.

9.6. 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: ECDH25519;

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

Algorithm Usage Locations: alg;

Change Controller: IANA;

Reference: Section 4.1 of this specification (for key derivation, see Section 5 of [SP-800-56c]);

Algorithm Analysis Documents: Section 4.1 of this specification (for key derivation, see Section 5 of [SP-800-56c]).

10. 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 and Vasily Nikolaev for their careful reviews.

11. References

11.1. 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-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.

[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>.

[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>.

[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>.

[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>.

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

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

[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.

11.2. Informative References

[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.

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

[HW-ECC] W.P. Liu, "How to Use the Kinets LTC ECC HW to Accelerate Curve25519 (version 7)", NXP, https://community.nxp.com/docs/DOC-330199, April 2017.

[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.

[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.

[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

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 "secant-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix C.1 for details of the group operation.

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 "secant-and-tangent" rule, where the point at infinity serves as the identity element. See Appendix C.2 for details of the group operation.

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.

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

Appendix B. Elliptic Curve Nomenclature and Finite Fields

B.1. Elliptic Curve Nomenclature

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.) If (P) has cardinality l, then l is called the order of P. 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. A point of order dividing h is said to be in the small subgroup. For curves of prime order, this small subgroup is the singleton set, consisting of only the identity element O. If a point is not in the small subgroup, it has order at least n.

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.

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.

In this document, a quadratic twist of a curve E defined over a field GF(q) is a curve E' related to E, 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 of this curve can be expressed using the same curve model, although (naturally) with its own curve parameters. Two curves E and E' defined over a field GF(q) are said to be isogenous if these have the same order and are said to be isomorphic if these have the same group structure. Note that isomorphic curves have necessarily the same order and are, thus, a special type of isogenous curves. Further details are out of scope.

Weierstrass curves 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 an odd prime power, is defined as follows.

If 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.

If q=p^m and 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, we always pick f(z):=z, so that the definions of GF(p) and GF(p^1) above coincide. If m>1, then GF(q) is called a (nontrivial) extension field over GF(p). The number p is called the characteristic of GF(q).

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 and inverses in GF(q) - if these exist - see Appendix L.1 and Appendix L.2, respectively.

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 J). Readers not interested in this, could simply view all fields as prime fields.

Appendix C. Elliptic Curve Group Operations

C.1. Group Law 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.

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=k*P=(X1, Y1), and P2=(k+1)*P=(X2, Y2) are distinct affine points of the Weierstrass curve W_{a,b} 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, as

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

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). Further details are out of scope.

C.2. Group Law 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.

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=k*P=(u1, v1), and P2=(k+1)*P=(u2, v2) are distinct affine points of the Montgomery curve M_{A,B} 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, as

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

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 one or two). Further details are out of scope.

C.3. **Group Law 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.

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=k*P=(x1, y1), and P2=(k+1)*P=(x2, y2) are affine points of the twisted Edwards curve E_{a,d} 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, as

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

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). Further details are out of scope.

Appendix D. Relationship 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}. 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}.

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/B+A/(3*B),v/B) of W_{a,b}. Note that not all Weierstrass curves can be injectively 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 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 on 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 curve 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

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.

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 (p+A)/3 for the isomorphic mapping and a shift of -(p+A)/3 for its inverse, where delta=(p+A)/3 is the element of GF(p) defined by

```
delta  19298681539552699237261830834781317975544997444273427339909597
    334652188435537

    (=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)

    51042569399160536130206135233146329284152202253034631822681833788
    666877215207

    (=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*(3*X-A)/(3*Y), (3*X-A-3)/(3*X-A+3)) 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   723700557733226221397318656304299424085711635937990760600195093828
    5454250989

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

h1  4

n1  144740111546645244279463731260859884816032634476503257978604941254
    07373907997

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

Montgomery curve-specific parameters (for Curve25519):

```
A   486662
```

B 1

Gu 9 (=0x9)

Gv 14781619447589544791020593568409986887264606134616475288964881837
755586237401

(=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)

(=370957059346694393431380835087545651895421138798432190163887855
33085940283555)

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

Gx 15112221349535400772501151409588531511454012693041857206046113283
949847762202

(=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 19298681539552699237261830834781317975544997444273427339909597334
573241639236

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

b 55751746669818908907645289078257140818241103727901012315294400837
956729358436

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

   GX  19298681539552699237261830834781317975544997444273427339909597334
       652188435546

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

   GY  14781619447589544791020593568409986887264606134616475288964881837
       755586237401

       (=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, which are known to allow for more efficient implementations of addition laws. 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 this mapping 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, which is known to allow for more efficient implementations of addition laws. 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, which is 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.

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. The inverse mapping from W_{a',b'} to W_{a,b} is again an 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 scope of this document.

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

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 H.1 and where t is the element of GF(p) defined by

```
t   35728133398289175649586938605660542688691615699169662967154525084
    644181596229

    (=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 H.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.3). 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	12102640281269758552371076649779977768474709596484288167752775713 178787220689 (=0x1ac1da05 b55bc146 33bd39e4 7f94302e f19843dc f669916f 6a5dfd01 65538cd1)
G2X	10770553138368400518417020196796161136792368198326337823149502681 097436401658 (=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 ffffffea)

   b   29689592517550930188872794512874050362622433571298029721775200646
       451501277098

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

   G3X 53837179229940872434942723257480777370451127212339198133697207846
       219400243292

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

   G3Y 69548073091100184414402055529279970392514867422855141773070804184
       60388229929

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

Appendix H. 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 H.1 (for the isogeny) and in Appendix H.2 (for the dual isogeny). For each polynomial in variable x, the coefficients are tabulated as sequence of coefficients of x^0, x^1, x^2, ..., in hexadecimal format.

H.1. Isogeny Parameters

H.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 0xfb3529b0d4b9ce2d70760f33e8ce997a58999718e9277caf48623d27ae6a788

   30 0x4352604bffd0c7d7a9ed898a2c6e7cf2512ffb89407271ba1f2c2d0ead8cc5aa

   31 0x6667697b29785fb6f0bd5e04d828991a5fe525370216f347ec767a26e7aac936

   32 0x9fc950b083c56dbd989badf9887255e203c879f123a7cb28901e50aea6d64dc

   33 0x41e51b51b5caadd1c15436bbf37596a1d7288a5f495d6b5b1ae66f8b2942b31d

   34 0x73b59fec709aa1cabd429e981c6284822a8b7b07620c831ab41fd31d5cf7430

   35 0x67e9b88e9a1bfbc2554107d67d814986f1b09c3107a060cba21c019a2d5dc848

   36 0x6881494a1066ca176c5e174713786040affb4268b19d2abf28ef4293429f89c1

   37 0x5f4d30502ff1e1ccd624e6f506569454ab771869d7483e26afc09dea0c5ccd3d

   38 0x2a814cfc5859bca51e539c159955cbe729a58978b52329575d09bc6c3bf97ad

   39 0x1313c8aaae20d6f4397f0d8b19e52cfcdf8d8e10fba144aec1778fd10ddf4e9c

   40 0x7008d38f434b98953a996d4cc79fcbef9502411dcdf92005f725cea7ce82ad47

   41 0x5a74d1296aaaa245ffb848f434531fa3ba9e5cb9098a7091d36c2777d4cf5a13

   42 0x4bd3b700606397083f8038177bdaa1ac6edbba0447537582723cae0fd29341a9

   43 0x573453fb2b093016f3368356c786519d54ed05f5372c01723b4da520597ec217

   44 0x77f5c605bdb3a30d7d9c8840fce38650910d4418eed707a212c8927f41c2c812

   45 0x16d6b9f7ff57ca32350057de1204cc6d69d4ef1b255dfef8080118e2fef6ace3

   46 0x34e8595832a4021f8b5744014c6b4f7da7df0d0329e8b6b4d44c8fadad6513b7

   47 0x1
```

H.1.2. Coefficients of v(x)

0	0xf9f5eb7134e6f8dafa30c45afa58d7bfc6d4e3ccbb5de87b562fd77403972b2
1	0x36c2dcd9e88f0d2d517a15fc453a098bbbb5a05eb6e8da906fae418a4e1a13f7
2	0xb40078302c24fa394a834880d5bf46732ca1b4894172fb7f775821276f558b3
3	0x53dd8e2234573f7f3f7df11e90a7bdd7b75d807f9712f521d4fb18af59aa5f26
4	0x6d4d7bb08de9061988a8cf6ff3beb10e933d4d2fbb8872d256a38c74c8c2ceda
5	0x71bfe5831b30e28cd0fbe1e9916ab2291c6beacc5af08e2c9165c632e61dd2f5
6	0x7c524f4d17ff2ee88463da012fc12a5b67d7fb5bd0ab59f4bbf162d76be1c89c
7	0x758183d5e07878d3364e3fd4c863a5dc1fe723f48c4ab4273fc034f5454d59a4
8	0x1eb41ef2479444ecdccbc200f64bde53f434a02b6c3f485d32f14da6aa7700e1
9	0x1490f3851f016cc3cf8a1e3c16a53317253d232ed425297531b560d70770315c
10	0x9bc43131964e46d905c3489c9d465c3abbd26eab9371c10e429b36d4b86469c
11	0x5f27c173d94c7a413a288348d3fc88daa0bcf5af8f436a47262050f240e9be3b
12	0x1d20010ec741aaa393cd19f0133b35f067adab0d105babe75fe45c8ba2732ceb
13	0x1b3c669ae49b86be2f0c946a9ff6c48e44740d7d9804146915747c3c025996a
14	0x24c6090f79ec13e3ae454d8f0f98e0c30a8938180595f79602f2ba013b3c10db
15	0x4650c5b5648c6c43ac75a2042048c699e44437929268661726e7182a31b1532f
16	0x957a835fb8bac3360b5008790e4c1f3389589ba74c8e8bf648b856ba7f22ba5
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 0x334ed41325f3724ff8becbf2b3443fea6d30fa543d1ca13188aceb2bdaf5f4e

   38 0x70e629c95a94e8e1b3974acb25e18ba42f8d5991786f0931f650c283adfe82fd

   39 0x738a625f4c62d3d645f1274e09ab344e72d441f3c0e82989d3e21e19212f23f3

   40 0x7093737294b29f21522f5664a9941c9b476f75d443b647bd2c777040bcd12a6a

   41 0xa996bad5863d821ccb8b89fa329ddbe5317a46bcb32552db396bea933765436

   42 0x2da237e3741b75dd0264836e7ef634fc0bc36ab187ebc790591a77c257b06f53

   43 0x1902f3daa86fa4f430b57212924fdc9e40f09e809f3991a0b3a10ab186c50ee5

   44 0x12baffec1bf20c921afd3cdf67a7f1d87c00d5326a3e5c83841593c214dadcb1

   45 0x6460f5a68123cb9e7bc1289cd5023c0c9ccd2d98eea24484fb3825b59dcd09aa

   46 0x2c7d63a868ffc9f0fd034f821d84736c5bc33325ce98aba5f0d95fef6f230ec8
```

```
   47 0x756e0063349a702db7406984c285a9b6bfba48177950d4361d8efa77408dc860

   48 0x37f3e30032b21e0279738e0a2b689625447831a2ccf15c638672da9aa7255ae

   49 0x1107c0dbe15d6ca9e790768317a40bcf23c80f1841f03ca79dd3e3ef4ea1ae30

   50 0x61ff7f25721d6206041c59a788316b09e05135a2aad94d539c65daa68b302cc2

   51 0x5dbfe346cbd0d61b9a3b5c42ec0518d3ae81cabcc32245060d7b0cd982b8d071

   52 0x4b6595e8501e9ec3e75f46107d2fd76511764efca179f69196eb45c0aa6fade3

   53 0x72d17a5aa7bd8a2540aa9b02d9605f2a714f44abfb4c35d518b7abc39b477870

   54 0x658d8c134bac37729ec40d27d50b637201abbf1ab4157316358953548c49cf22

   55 0x36ac53b9118581ace574d5a08f9647e6a916f92dda684a4dbc405e2646b0243f

   56 0x1917a98f387d1e323e84a0f02d53307b1dd949e1a27b0de14514f89d9c0ef4b6

   57 0x21573434fde7ce56e8777c79539479441942dba535ade8ecb77763f7eb05d797

   58 0xe0bf482dc40884719bea5503422b603f3a8edb582f52838caa6eaab6eeac7ef

   59 0x3b0471eb53bd83e14fbc13928fe1691820349a963be8f7e9815848a53d03f5eb

   60 0x1e92cb067b24a729c42d3abb7a1179c577970f0ab3e6b0ce8d66c5b8f7001262

   61 0x74ea885c1ebed6f74964262402432ef184c42884fceb2f8dba3a9d67a1344dd7

   62 0x433ebce2ce9b0dc314425cfc2b234614d3c34f2c9da9fff4fdddd1ce242d035b

   63 0x33ac69e6be858dde7b83a9ff6f11de443128b39cec6e410e8d3b570e405ff896

   64 0xdab71e2ae94e6530a501ed8cf3df26731dd1d41cd81578341e12dca3cb71aa3

   65 0x537f58d52d18ce5b1d5a6bd3a420e796e64173491ad43dd4d1083a7dcc7dd201

   66 0x49c2f6afa93fdcc4e0f8128a8b06da4c75049be14edf3e103821ab604c60f8ae

   67 0x10a333eabd6135aeaa3f5f5f7e73d102e4fd7e4bf0902fc55b00da235fa1ad08

   68 0xf5c86044bf6032f5102e601f2a0f73c7bce9384bedd120f3e72d78484179d9c

   69 0x1
```

H.1.3. Coefficients of w(x)

0	0x3da24d42421264f30939ff00203880f2b017eb3fecf8933ae61e18df8c8ba116
1	0x457f20bc393cdc9a66848ce174e2fa41d77e6dbae05a317a1fb6e3ae78760f8
2	0x7f608a2285c480d5c9592c435431fae94695beef79d770bb6d029c1d10a53295
3	0x3832accc520a485100a0a1695792465142a5572bed1b2e50e1f8f662ac7289bb
4	0x2df1b0559e31b328eb34beedd5e537c3f4d7b9befb0749f75d6d0d866d26fbaa
5	0x25396820381d04015a9f655ddd41c74303ded05d54a7750e2f58006659adda28
6	0x6fa070a70ca2bc6d4d0795fb28d4990b2cc80cd72d48b603a8ac8c8268bef6a6
7	0x27f488578357388b20fbc7503328e1d10de602b082b3c7b8ceb33c29fea7a0d2
8	0x15776851a7cabcfe84c632118306915c0c15c75068a47021968c7438d46076e6
9	0x101565b08a9af015c172fb194b940a4df25c4fb1d85f72d153efc79131d45e8f
10	0x196b0ffbf92f3229fea1dac0d74591b905ccaab6b83f905ee813ee8449f8a62c
11	0x1f55784691719f765f04ee9051ec95d5deb42ae45405a9d87833855a6d95a94
12	0x628858f79cca86305739d084d365d5a9e56e51a4485d253ae3f2e4a379fa8aff
13	0x4a842dcd943a80d1e6e1dab3622a8c4d390da1592d1e56d1c14c4d3f72dd01a5
14	0xf3bfc9cb17a1125f94766a4097d0f1018963bc11cb7bc0c7a1d94d65e282477
15	0x1c4bd70488c4882846500691fa7543b7ef694446d9c3e3b4707ea2c99383e53c
16	0x2d7017e47b24b89b0528932c4ade43f09091b91db0072e6ebdc5e777cb215e35
17	0x781d69243b6c86f59416f91f7decaca93eab9cdc36a184191810c56ed85e0fdc
18	0x5f20526f4177357da40a18da054731d442ad2a5a4727322ba8ed10d32eca24fb
19	0x33e4cab64ed8a00d8012104fe8f928e6173c428eff95bbbe569ea46126a4f3cd
20	0x50555b6f07e308d33776922b6566829d122e19b25b7bbacbb0a4b1a7dc40192
21	0x533fa4bf1e2a2aae2f979065fdbb5b667ede2f85543fddbba146aa3a4ef2d281
22	0x5a742cac1952010fc5aba200a635a7bed3ef868194f45b5a6a2647d6d6b289d2

23	0x1

H.2. Dual Isogeny Parameters

H.2.1. Coefficients of u'(x)

0	0xf0eddb584a20aaac8f1419efdd02a5cca77b21e4cfae78c49b5127d98bc5882
1	0x7115e60d44a58630417df33dd45b8a546fa00b79fea3b2bdc449694bade87c0a
2	0xb3f3a6f3c445c7dc1f91121275414e88c32ff3f367ba0edad4d75b7e7b94b65
3	0x1eb31bb333d7048b87f2b3d4ec76d69035927b41c30274368649c87c52e1ab30
4	0x552c886c2044153e280832264066cce2a7da1127dc9720e2a380e9d37049ac64
5	0x4504f27908db2e1f5840b74ae42445298755d9493141f5417c02f04d47797dda
6	0x82c242cce1eb19698a4fa30b5affe64e5051c04ae8b52cb68d89ee85222e628
7	0x480473406add76cf1d77661b3ff506c038d9cdd5ad6e1ea41969430bb876d223
8	0x25f47bb506fba80c79d1763365fa9076d4c4cb6644f73ed37918074397e88588
9	0x10f13ed36eab593fa20817f6bb70cac292e18d300498f6642e35cbdf772f0855
10	0x7d28329d695fb3305620f83a58df1531e89a43c7b3151d16f3b60a8246c36ade
11	0x2c5ec8c42b16dc6409bdd2c7b4ffe9d65d7209e886badbd5f865dec35e4ab4a
12	0x7f4f33cd50255537e6cde15a4a327a5790c37e081802654b56c956434354e133
13	0x7d30431a121d9240c761998cf83d228237e80c3ef5c7191ec9617208e0ab8cec
14	0x4d2a7d6609610c1deed56425a4615b92f70a507e1079b2681d96a2b874cf0630
15	0x74676df60a9906901d1dc316c639ff6ae0fcdb02b5571d4b83fc2eedcd2936a8
16	0x22f8212219aca01410f06eb234ed53bd5b8fbe7c08652b8002bcd1ea3cdae387
17	0x7edb04449565d7c566b934a87fadade5515f23bda1ce25daa19fff0c6a5ccc2f
18	0x106ef71aa3aa34e8ecf4c07a67d03f0949d7d015ef2c1e32eb698dd3bec5a18c
19	0x17913eb705db126ac3172447bcd811a62744d505ad0eea94cfcfdde5ca7428
20	0x2cc793e6d3b592dcf5472057a991ff1a5ab43b4680bb34c0f5faffc5307827c1

```
   21 0x6dafcc0b16f98300cddb5e0a7d7ff04a0e73ca558c54461781d5a5ccb1ea0122

   22 0x7e418891cf222c021b0ae5f5232b9c0dc8270d4925a13174a0f0ac5e7a4c8045

   23 0x76553bd26fecb019ead31142684789fea7754c2dc9ab9197c623f45d60749058

   24 0x693efb3f81086043656d81840902b6f3a9a4b0e8f2a5a5edf5ce1c7f50a3898e

   25 0x46c630eac2b86d36f18a061882b756917718a359f44752a5caf41be506788921

   26 0x1dcfa01773628753bc6f448ac11be8a3bffa0011b9284967629b827e064f614

   27 0x8430b5b97d49b0938d1f66ecb9d2043025c6eec624f8f02042b9621b2b5cb19

   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 0xdc29a73b97f08aa8774911474e651130ed364e8d8cffd4a80dee633aacecc47

   41 0x4e7eb8584ae4de525389d1e9300fc4480b3d9c8a5a45ecfbe33311029d8f6b99

   42 0x6c3cba4aa9229550fa82e1cfaee4b02f2c0cb86f79e0d412b8e32b00b7959d80

   43 0x5a9d104ae585b94af68eeb16b1349776b601f97b7ce716701645b1a75b68dcf3

   44 0x754e014b5e87af035b3d5fe6fb49f4631e32549f6341c6693c5172a6388e273e
```

```
   45 0x6710d8265118e22eaceba09566c86f642ab42da58c435083a353eaa12d866c39

   46 0x6e88ac659ce146c369f8b24c3a49f8dca547827250cf7963a455851cfc4f8d22

   47 0x971eb5f253356cd1fde9fb21f4a4902aa5b8d804a2b57ba775dc130181ae2e8
```

H.2.2. Coefficients of v'(x)

```
   0  0x43c9b67cc5b16e167b55f190db61e44d48d813a7112910f10e3fd8da85d61d3

   1  0x72046db07e0e7882ff3f0f38b54b45ca84153be47a7fd1dd8f6402e17c47966f

   2  0x1593d97b65a070b6b3f879fe3dc4d1ef03c0e781c997111d5c1748f956f1ffc0

   3  0x54e5fec076b8779338432bdc5a449e36823a0a7c905fd37f232330b026a143a0

   4  0x46328dd9bc336e0873abd453db472468393333fbf2010c6ac283933216e98038

   5  0x25d0c64de1dfe1c6d5f5f2d98ab637d8b39bcf0d886a23dabac18c80d7eb03ce

   6  0x3a175c46b2cd8e2b313dde2d5f3097b78114a6295f283cf58a33844b0c8d8b34

   7  0x5cf4e6f745bdd61181a7d1b4db31dc4c30c84957f63cdf163bee5e466a7a8d38

   8  0x639071c39b723eea51cfd870478331d60396b31f39a593ebdd9b1eb543875283

   9  0x7ea8f895dcd85fc6cb2b58793789bd9246e62fa7a8c7116936876f4d8dff869b

   10 0x503818acb535bcaacf8ad44a83c213a9ce83af7c937dc9b3e5b6efedc0a7428c

   11 0xe815373920ec3cbf3f8cae20d4389d367dc4398e01691244af90edc3e6d42b8

   12 0x7e4b23e1e0b739087f77910cc635a92a3dc184a791400cbceae056c19c853815

   13 0x145322201db4b5ec0a643229e07c0ab7c36e4274745689be2c19cfa8a702129d

   14 0xfde79514935d9b40f52e33429621a200acc092f6e5dec14b49e73f2f59c780d

   15 0x37517ac5c04dc48145a9d6e14803b8ce9cb6a5d01c6f0ad1b04ff3353d02d815

   16 0x58ae96b8eefe9e80f24d3b886932fe3c27aaea810fa189c702f93987c8c97854

   17 0x6f6402c90fa379096d5f436035bebc9d29302126e9b117887abfa7d4b3c5709a

   18 0x1dbdf2b9ec09a8defeb485cc16ea98d0d45c5b9877ff16bd04c0110d2f64961

   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 0xf6e6b3fc54c2d25871011b1499bb0ef015c6d0da802ae7eccf1d8c3fb73856c

   41 0xc1422c98b672414344a9c05492b926f473f05033b9f85b8788b4bb9a080053c

   42 0x19a8527de35d4faacb00184e0423962247319703a815eecf355f143c2c18f17f

   43 0x7812dc3313e6cf093da4617f06062e8e8969d648dfe6b5c331bccd58eb428383
```

```
   44 0x61e537180c84c79e1fd2d4f9d386e1c4f0442247605b8d8904d122ee7ef9f7be

   45 0x544d8621d05540576cfc9b58a3dab19145332b88eb0b86f4c15567c37205adf9

   46 0x11be3ef96e6e07556356b51e2479436d9966b7b083892b390caec22a117aa48e

   47 0x205cda31289cf75ab0759c14c43cb30f7287969ea3dc0d5286a3853a4d403187

   48 0x48d8fc6934f4f0a99f0f2cc59010389e2a0b20d6909bfcf8d7d0249f360acdc

   49 0x42cecc6d9bdca6d382e97fcea46a79c3eda2853091a8f399a2252115bf9a1454

   50 0x117d41b24f2f69cb3270b359c181607931f62c56d070bbd14dc9e3f9ab1432e

   51 0x7c51564c66f68e2ad4ce6ea0d68f920fafa375376709c606c88a0ed44207aa1e

   52 0x48f25191fc8ac7d9f21adf6df23b76ccbca9cb02b815acdbebfa3f4eddc71b34

   53 0x4fc21a62c4688de70e28ad3d5956633fc9833bc7be09dc7bc500b7fae1e1c9a8

   54 0x1f23f25be0912173c3ef98e1c9990205a69d0bf2303d201d27a5499247f06789

   55 0x3131495618a0ac4cb11a702f3f8bab66c4fa1066d0a741af3c92d5c246edd579

   56 0xd93fe40faa53913638e497328a1b47603cb062c7afc9e96278603f29fd11fd4

   57 0x6b348bc59e984c91d696d1e3c3cfae44021f06f74798c787c355437fb696093d

   58 0x65af00e73043edcb479620c8b48098b89809d577a4071c8e33e8678829138b8a

   59 0x5e62ffb032b2ddb06591f86a46a18effd5d6ecf3f129bb2bacfd51a3739a98b6

   60 0x62c974ef3593fc86f7d78883b8727a2f7359a282cbc0196948e7a793e60ce1a1

   61 0x204d708e3f500aad64283f753e7d9bab976aa42a4ca1ce5e9d2264639e8b1110

   62 0xa90f0059da81a012e9d0a756809fab2ce61cb45965d4d1513a06227783ee4ea

   63 0x39fa55971c9e833f61139c39e243d40869fd7e8a1417ee4e7719dd2dd242766f

   64 0x22677c1e659caa324f0c74a013921facf62d0d78f273563145cc1ddccfcc4421

   65 0x3468cf6df7e93f7ff1fe1dd7e180a89dec3ed4f72843b4ea8a8d780011a245b2

   66 0x68f75a0e2210f52a90704ed5f511918d1f6bcfcd26b462cc4975252369db6e9d

   67 0x6220c0699696e9bcab0fe3a80d437519bd2bdf3caef665e106b2dd47585ddd9f
```

```
   68 0x553ad47b129fb347992b576479b0a89f8d71f1196f83e5eaab5f533a1dd6f6d7

   69 0x239aef387e116ec8730fa15af053485ca707650d9f8917a75f22acf6213197df
```

H.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  0xd837dbbd1af32d04e2699cb026399c1928472aa1a7f0a1d3afd24bc9923456a

   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 0x1
```

Appendix I. 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 the 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) and then extend to all fields GF(q), where q is an odd prime power. We assume each finite field to be unambiguously defined.

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.

I.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^2+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, 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.

If the Weierstrass curve W_{a,b} has a point of order two, we extend the definition of the point compression function to all points of the curve, by associating the (non-affine) point at infinity O with the ordered pair compr(O):=(X,1), where (X,0) is any point of order two, and recover this point accordingly. (Note that this corresponds to the case alpha=0 and t=1 above.) In this case, the point at infinity O can be represented by any ordered pair (X, 1) of elements of GF(q), where (X,0) is any point of order two. Note that this ordered pair does not satisfy the defining equation of the curve in question.

I.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, 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.

I.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, 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). 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} (subject to the Note in Appendix C.3). Here, the identity element O:=(0,1) is associated with the compressed point compr(O):=(0,1).

Appendix J. Data Conversions

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). 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.

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. 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 we define these notions as well if t is outside the interval [0,l]: a t-prefix or t-postfix is the empty string if t is negative and 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.

An octet 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}. Note that the length of a string is defined in terms of the underlying alphabet, as are the operations in the previous paragraph.

J.1. Conversion between Bit Strings and Integers

There is a 1-1 correspondence between bit strings of length l and the 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.

J.2. Conversion between Octet Strings and Integers (OS2I, I2OS)

There is a 1-1 correspondence between octet strings of length l and the 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. 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.

J.3. Conversion between Octet Strings and Bit Strings (BS2OS, OS2BS)

There is a 1-1 correspondence between octet strings of length l and 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 J.1 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.

J.4. Conversion between Field Elements and Octet Strings (FE2OS, OS2FE)

There is a 1-1 correspondence between elements of a fixed finite field GF(q), where q=p^m and 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 J.2 above. Note that both the mapping from field elements to octet strings and the inverse mapping 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.

J.5. Conversion between Elements of Z mod n and Octet Strings (ZnE2OS, OS2ZnE)

There is a 1-1 correspondence between elements of a fixed set Z(n) of integers modulo n and integers in the interval [0,n), where each element x can be uniquely represented by the octet string corresponding to the integer x modulo n according to the mapping of Appendix J.2 above. Note that both the mapping from elements of Z(n) to octet strings and the inverse mapping 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 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.

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 n values are useful, e.g., when encoding the modulus n of RSA (or elements of any other non-prime set Z(n), for that matter).

J.6. 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), which we define both for bit strings and octet strings. For a bit string [octet string] X:=str(x_{l-1}, x_{l-2}, ..., x_1, x_0), its reverse is the bit string [octet string] X':=rev(X):=str(x_0, x_1, ..., x_{l-2}, x_{l-1}).

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.)

Note that elements of a prime field GF(p), where p is a 255-bit prime number, have a tight representation as a 32-byte 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 I) to be encoded in this bit position and, thereby, allows a compressed point and a field element of GF(p) to be represented by an octet string of the same length. This is called the squeezed point representation. Obviously, other representations (e.g., those of elements of Z(n)) may also have fixed bit values on certain positions, which can be used to squeeze-in additional information. Further details are out of scope.

Appendix K. Representation Examples Curve25519 Family Members

We present some examples of computations using the curves introduced in 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   45467544759954639344191351164156560595299236761702065033670739677
    691372543056

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

K.1. Example with Curve25519

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

```
u   53025657538808013645618620393754461319535915376830819974982289332
    088255623750

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

v 53327798092436462013048370302019946300826511459161905709144645521
233690313086

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

u1 42039618818474335439333192910143029294450651736166602435248528442
691717668056

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

v1 76981661982917351630937517222412729130882368858134322156485762195
67913357634

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

u2 34175116482377882355440137752573651838273760818624557524643126101
82464621878

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

v2 43046985853631671610553834968785204191967171967937842531656254539
962663994648

(=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 I.2 and Appendix J 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).

K.2. Example with Edwards25519

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

```
x   25301662348702136092602268236183361085863932475593120475382959053
    365387223252

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

y   54434749145175762798550436656748568411099702168121592090608501578
    942019473360

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

x1  42966967796585460733861724865699548279978730460766025087444502812
    416557284873

    (=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 I.3 and Appendix J 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 [RFC8032], [RFC8032] requires unique representations of all elements of GF(p).

K.3. Example with Wei25519

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

```
   X    14428294459702615171094958724191825368445920488283965295163094662
        783879239338

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

   Y    53327798092436462013048370302019946300826511459161905709144645521
        233690313086

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

```
X1  344225573936893696480953124058039334336065684761974775542933377333
    87341283644

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

Y1  769816619829173516309375172224127291308823688581343221564857621955
    67913357634

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

X2  227161931877904874728058446100386831593723735261358830923739099448
    834653057415

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

Y2  430469858536316716105538349687852041919671719679378425316562545399
    962663994648

    (=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 I.1 and Appendix J 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].

K.4. Example with Wei25519.2

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

```
X   17830493209951148331008014701079988862634531394137235438571836389
    227198459763

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

Y   21064492012933896105338241940477778461866060481408222122979836206
    137075789640

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

X1  65470988951686461979789632362377759464688342154017353834939203791
    39281908968
```

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

   Y1  51489590494292183562535790579480033229043271539297275888817125227
       35262330110

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

   X2  83741788501517200942826153677682120998854086551751663061374935388
       3494226693

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

   Y2  42567334190622848157611574766896093933050043101247319937794684825
       168161540336

       (=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 I.1 and Appendix J for further detail on (squeezed) point compression.

K.5. Example with Wei25519.-3

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

```
   X   14780197759513083469009623947734627174363231692126610860256057394
       455099634096

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

```
   Y   45596733430378470319805536538617129933663237960146030424392249401
       952949482817

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

   X1  47362979975244556396292400751828272600887612546997532158738958926
       60745725532

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

   Y1  30318112837157047703426636957515037640997356617656007157255559136
       153389790354

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

   X2  23778942085873786433506063022059853212880296499622328201295446580
       293591664363

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

   Y2  44846366394651736248316749170687053272682847823018287439056537991
       969511150494

       (=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 I.1 and Appendix J for further detail on (squeezed) point compression.

Appendix L. Auxiliary Functions

L.1. Square Roots in GF(q)

Square roots are easy to compute in GF(q) if q = 3 (mod 4) (see Appendix L.1.1) or if q = 5 (mod 8) (see Appendix L.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).

L.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. 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).

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

If y is a nonzero element of GF(q) and z:=y^{z-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.

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

L.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).

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

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.

Author's Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com