CFRG Y. Sakemi, Ed.
Internet-Draft S. Kanno
Intended status: Informational GMO CONNECT Inc.
Expires: 6 May 2026 R. Wahby
Stanford University
2 November 2025

Pairing-Friendly Curves
draft-irtf-cfrg-pairing-friendly-curves-12

Abstract

Pairing-based cryptography, a subfield of elliptic curve cryptography, has received attention due to its flexible and practical functionality. Pairings are special maps defined using elliptic curves and it can be applied to construct several cryptographic protocols such as identity-based encryption, attribute-based encryption, and so on. At CRYPTO 2016, Kim and Barbulescu proposed an efficient number field sieve algorithm named exTNFS for the discrete logarithm problem in a finite field. Several types of pairing-friendly curves such as Barreto-Naehrig curves are affected by the attack. In particular, a Barreto-Naehrig curve with a 254-bit characteristic was adopted by a lot of cryptographic libraries as a parameter of 128-bit security, however, it ensures no more than the 100-bit security level due to the effect of the attack. In this memo, we list the security levels of certain pairing-friendly curves, and motivate our choices of curves. First, we summarize the adoption status of pairing-friendly curves in standards, libraries and applications, and classify them in the 128-bit, 192-bit, and 256-bit security levels. Then, from the viewpoints of "security" and "widely used", we select the recommended pairing-friendly curves considering exTNFS.

Status of This Memo

This Internet-Draft will expire on 6 May 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Pairing-based Cryptography

Elliptic curve cryptography is an important area in currently deployed cryptography. The cryptographic algorithms based on elliptic curve cryptography, such as the Elliptic Curve Digital Signature Algorithm (ECDSA), are widely used in many applications.

Pairing-based cryptography, a subfield of elliptic curve cryptography, has attracted much attention due to its flexible and practical functionality. Pairings are special maps defined using elliptic curves. Pairings are fundamental in the construction of several cryptographic algorithms and protocols such as identity-based encryption (IBE), attribute-based encryption (ABE), authenticated key exchange (AKE), short signatures, and so on. Several applications of pairing-based cryptography are currently in practical use.

As the importance of pairings grows, elliptic curves where pairings are efficiently computable are studied and the special curves called pairing-friendly curves are proposed.

1.2. Applications of Pairing-based Cryptography

Several applications using pairing-based cryptography have already been standardized and deployed. We list here some examples of applications available in the real world.

IETF published RFCs for pairing-based cryptography such as Identity-Based Cryptography [RFC5091], Sakai-Kasahara Key Encryption (SAKKE) [RFC6508], and Identity-Based Authenticated Key Exchange (IBAKE) [RFC6539]. SAKKE is applied to Multimedia Internet KEYing (MIKEY) [RFC6509] and used in 3GPP [SAKKE].

Pairing-based key agreement protocols are standardized in ISO/IEC [ISOIEC11770-3]. In [ISOIEC11770-3], a key agreement scheme by Joux [Joux00], identity-based key agreement schemes by Smart-Chen-Cheng [CCS07] and Fujioka-Suzuki-Ustaoglu [FSU10] are specified.

MIRACL implements M-Pin, a multi-factor authentication protocol [M-Pin]. The M-Pin protocol includes a type of zero-knowledge proof, where pairings are used for its construction.

The Trusted Computing Group (TCG) specified the Elliptic Curve Direct Anonymous Attestation (ECDAA) in the specification of a Trusted Platform Module (TPM) [TPM]. ECDAA is a protocol for proving the attestation held by a TPM to a verifier without revealing the attestation held by that TPM. Pairings are used in the construction of ECDAA. FIDO Alliance [FIDO] and W3C [W3C] also published an ECDAA algorithm similar to TCG.

Intel introduced Intel Enhanced Privacy ID (EPID) that enables remote attestation of a hardware device while preserving the privacy of the device as part of the functionality of Intel Software Guard Extensions (SGX) [EPID]. They extended TPM ECDAA to realize such functionality. A pairing-based EPID was proposed [BL10] and distributed along with Intel SGX applications.

Zcash implemented their own zero-knowledge proof algorithm named Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARKs) [Zcash]. zk-SNARKs are used for protecting the privacy of transactions of Zcash. They use pairings to construct zk-SNARKs.

Cloudflare introduced Geo Key Manager [Cloudflare] to restrict distribution of customers’ private keys to a subset of their data centers. To achieve this functionality, ABE is used, and pairings take a role as a building block. In addition, Cloudflare published a new cryptographic library, the Cloudflare Interoperable, Reusable Cryptographic Library (CIRCL) [CIRCL] in 2019. They plan to include securely implemented subroutines for pairing computations on certain secure pairing-friendly curves in CIRCL.

Currently, Boneh-Lynn-Shacham (BLS) signature schemes are being standardized [I-D.boneh-bls-signature] and utilized in several blockchain projects such as Ethereum [Ethereum], Algorand [Algorand], Chia Network [Chia], and DFINITY [DFINITY]. The aggregation functionality of BLS signatures is effective for their applications of decentralization and scalability.

1.3. Motivation and Contribution

At CRYPTO 2016, Kim and Barbulescu proposed an efficient number field sieve (NFS) algorithm for the discrete logarithm problem in a finite field GF(p^k) [KB16]. The attack improves the polynomial selection that is the first step in the number field sieve algorithm for discrete logarithms in GF(p^k). The idea is applicable when the embedding degree k is a composite that satisfies k = i*j (gcd (i, j) = 1, i, j> 1). The basic idea is based on the equality GF(p^k) = (GF(p^i)^j) and one of the improvement for reducing the amount of cost for solving the discrete logarithm problem is using sub-field calculation. Several types of pairing-friendly curves such as Barreto-Naehrig curves (BN curves)[BN05] and Barreto-Lynn-Scott curves (BLS curves)[BLS02] are affected by the attack, since a pairing-friendly curve suitable for cryptographic applications requires that the discrete logarithm problem is sufficiently difficult. Please refer to [KB16] for detailed ideas and calculation algorithms of the attack by Kim. In particular, BN254, which is a BN curve with a 254-bit characteristic effective for pairing calculations, was adopted by a lot of cryptographic libraries as a parameter of the 128-bit security level, however, BN254 ensures no more than the 100-bit security level due to the effect of the attack, where the security levels described in this memo correspond to the security strength of NIST recommendation [NIST].

To resolve this effect immediately, several research groups and implementers re-evaluated the security of pairing-friendly curves and they respectively proposed various curves that are secure against the attack [BD18] [BLS12_381].

In this memo, we list the security levels of certain pairing-friendly curves, and motivate our choices of curves. First, we summarize the adoption status of pairing-friendly curves in international standards, libraries and applications, and classify them in the 128-bit, 192-bit, and 256-bit security levels. Then, from the viewpoints of "security" and "widely used", pairing-friendly curves corresponding to each security level are selected in accordance with the security evaluation by Barbulescu and Duquesne [BD18].

As a result, we recommend the BLS curve with 381-bit characteristic of embedding degree 12 and the BN curve with the 462-bit characteristic for the 128-bit security level, and the BLS curves of embedding degree 48 with the 581-bit characteristic for the 256-bit security level. This memo shows their specific test vectors.

1.4. Requirements Terminology

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

2. Preliminaries

2.1. Elliptic Curves

Let p be a prime number and q = p^n for a natural number n > 0, where p at least 5. Let GF(q) be a finite field. The curve defined by the following equation E is called an elliptic curve:

    E : y^2 = x^3 + a * x + b,

and a and b in GF(q) satisfy the discriminant inequality 4 * a^3 + 27 * b^2 != 0 mod q. This is called the Weierstrass normal form of an elliptic curve.

A solution (x,y) to the equation E can be thought of as a point on the corresponding curve. For a natural number k, we define the set of (GF(q^k))-rational points of E, denoted by E(GF(q^k)), to be the set of all solutions (x,y) in GF(q^k), together with a 'point at infinity' O_E, which is defined to lie on every vertical line passing through the curve E.

The set E(GF(q^k)) forms a group under a group law that can be defined geometrically as follows. For P and Q in E(GF(q^k)) define P + Q to be the reflection around the x-axis of the unique third point R of intersection of the straight line passing through P and Q with the curve E. If the straight line is tangent to E, we say that it passes through that point twice. The identity of this group is the point at infinity O_E. We also define scalar multiplication [K]P for a positive integer K as the point P added to itself (K-1) times. Here, [0]P becomes the point at infinity O_E and the relation [-K]P = -([K]P) is satisfied.

2.2. Pairings

A pairing is a bilinear map defined on two subgroups of rational points of an elliptic curve. Examples include the Weil pairing, the Tate pairing, the optimal Ate pairing [Ver09], and so on. The optimal Ate pairing is considered to be the most efficient to compute and is the one that is most commonly used for practical implementation.

Let E be an elliptic curve defined over a prime field GF(p). Let k be the minimum integer for which r is a divisor of p^k - 1; this is called the embedding degree of E over GF(p). Let G_1 be a cyclic subgroup of E(GF(p)) of order r, there also exists a cyclic subgroup of E(GF(p^k)) of order r, define this to be G_2. Let d be a divisor of k and E' be an elliptic curve defined over GF(p^(k/d)). If an isomorphism from E' to E(GF(p^k)) exists, then E' is called the twist of E. It can sometimes be convenient for efficiency to do the computations of G_2 in the twist E', and so consider G_2 to instead be a subgroup of E'. Let G_T be an order r subgroup of the multiplicative group (GF(p^k))^*; this exists by definition of k.

A pairing is defined as a bilinear map e: (G_1, G_2) -> G_T satisfying the following properties:

1. Bilinearity: for any S in G_1, T in G_2, and integers K and L, e([K]S, [L]T) = e(S, T)^{K * L}.

2. Non-degeneracy: for any T in G_2, e(S, T) = 1 if and only if S = O_E. Similarly, for any S in G_1, e(S, T) = 1 if and only if T = O_E.

In applications, it is also necessary that for any S in G_1 and T in G_2, this bilinear map is efficiently computable.

We define some of the terminology used in this memo as follows:

GF(p): a finite field with characteristic p.

GF(p^k): an extension field of degree k.

(GF(p))*: a multiplicative group of GF(p).

(GF(p^k))*: a multiplicative group of GF(p^k).

b: a primitive element of the multiplicative group (GF(p))^*.

O_E: the point at infinity over an elliptic curve E.

E(GF(p^k)): the group of GF(p^k)-rational points of E.

#E(GF(p^k)): the number of GF(p^k)-rational points of E.

r: the order of G_1 and G_2.

BP: a point in G_1. (The 'base point' of a cyclic subgroup of G_1)

h: the cofactor h = #E(GF(p)) / r, where gcd(h, r)=1.

2.3. Barreto-Naehrig Curves

A BN curve [BN05] is a family of pairing-friendly curves proposed in 2005. A pairing over BN curves constructs optimal Ate pairings.

A BN curve is defined by elliptic curves E and E' parameterized by a well-chosen integer t. E is defined over GF(p), where p is a prime number and at least 5, and E(GF(p)) has a subgroup of prime order r. The characteristic p and the order r are parameterized by

```
p = 36 * t^4 + 36 * t^3 + 24 * t^2 + 6 * t + 1
r = 36 * t^4 + 36 * t^3 + 18 * t^2 + 6 * t + 1
```

for an integer t.

The elliptic curve E has an equation of the form E: y^2 = x^3 + b, where b is a primitive element of the multiplicative group (GF(p))^* of order (p - 1).

In the case of BN curves, we can use twists of the degree 6. If m is an element that is neither a square nor a cube in an extension field GF(p^2), the twist E' of E is defined over an extension field GF(p^2) by the equation E': y^2 = x^3 + b' with b' = b / m or b' = b * m. BN curves are called D-type if b' = b / m, and M-type if b' = b * m. The embedding degree k is 12.

A pairing e is defined by taking G_1 as a subgroup of E(GF(p)) of order r, G_2 as a subgroup of E'(GF(p^2)), and G_T as a subgroup of a multiplicative group (GF(p^12))^* of order r.

2.4. Barreto-Lynn-Scott Curves

A BLS curve [BLS02] is a another family of pairing-frinedly curves proposed in 2002. Similar to BN curves, a pairing over BLS curves constructs optimal Ate pairings.

A BLS curve is defined by elliptic curves E and E' parameterized by a well-chosen integer t. E is defined over a finite field GF(p) by an equation of the form E: y^2 = x^3 + b, and its twist E': y^2 = x^3 + b', is defined in the same way as BN curves. In contrast to BN curves, E(GF(p)) does not have a prime order. Instead, its order is divisible by a large parameterized prime r and denoted by h * r with cofactor h. The pairing is defined on the r-torsion points. In the same way as BN curves, BLS curves can be categorized as D-type and M-type.

BLS curves vary in accordance with different embedding degrees. In this memo, we deal with the BLS12 and BLS48 families with embedding degrees 12 and 48 with respect to r, respectively.

In BLS curves, parameters p and r are given by the following equations:

```
BLS12:
    p = (t - 1)^2 * (t^4 - t^2 + 1) / 3 + t
    r = t^4 - t^2 + 1
BLS48:
    p = (t - 1)^2 * (t^16 - t^8 + 1) / 3 + t
    r = t^16 - t^8 + 1
```

for a well chosen integer t where t must be 1 (mod 3).

A pairing e is defined by taking G_1 as a subgroup of E(GF(p)) of order r, G_2 as an order r subgroup of E'(GF(p^2)) for BLS12 and of E'(GF(p^8)) for BLS48, and G_T as an order r subgroup of a multiplicative group (GF(p^12))^* for BLS12 and of a multiplicative group (GF(p^48))^* for BLS48.

2.5. Representation Convention for an Extension Field

Pairing-friendly curves use a tower of some extension fields. In order to encode an element of an extension field, focusing on interoperability, we adopt the representation convention shown in Appendix J.4 of [I-D.ietf-lwig-curve-representations] as a standard and effective method. Note that the big-endian encoding is used for an element in GF(p) which follows to mcl [mcl], ISO/IEC 15946-5 [ISOIEC15946-5] and etc.

Let GF(p) be a finite field of characteristic p and GF(p^d) = GF(p)(i) be an extension field of GF(p) of degree d.

For an element s in GF(p^d) such that s = s_0 + s_1 * i + ... + s_{d - 1} * i^{d - 1} where s_0, s_1, ... , s_{d - 1} in the basefield GF(p), s is represented as octet string by oct(s) = s_0 || s_1 || ... || s_{d - 1}.

Let GF(p^d') = GF(p^d)(j) be an extension field of GF(p^d) of degree d' / d.

For an element s' in GF(p^d') such that s' = s'_0 + s'_1 * j + ... + s'_{d' / d - 1} * j^{d' / d - 1} where s'_0, s'_1, ..., s'_{d' / d - 1} in the basefield GF(p^d), s' is represented as integer by oct(s') = oct(s'_0) || oct(s'_1) || ... || oct(s'_{d' / d - 1}), where oct(s'_0), ... , oct(s'_{d' / d - 1}) are octet strings encoded by above convention.

In general, one can define encoding between integer and an element of any finite field tower by inductively applying the above convention.

The parameters and test vectors of extension fields described in this memo are encoded by this convention and represented in an octet stream.

When applications communicate elements in an extension field, using the compression method [MP04] may be more effective. In that case, care for interoperability must be taken.

3. Security of Pairing-Friendly Curves

3.1. Evaluating the Security of Pairing-Friendly Curves

The security of pairing-friendly curves is evaluated by the hardness of the following discrete logarithm problems:

* The elliptic curve discrete logarithm problem (ECDLP) in G_1 and G_2

* The finite field discrete logarithm problem (FFDLP) in G_T

There are other hard problems over pairing-friendly curves used for proving the security of pairing-based cryptography. Such problems include the computational bilinear Diffie-Hellman (CBDH) problem, the bilinear Diffie-Hellman (BDH) problem, the decision bilinear Diffie-Hellman (DBDH) problem, the gap DBDH problem, etc. [ECRYPT]. Almost all of these variants are reduced to the hardness of discrete logarithm problems described above and are believed to be easier than the discrete logarithm problems.

Although it would be sufficient to attack any of these problems to attack pairing-based crytography, the only known attacks thus far attack the discrete logarithm problem directly, so we focus on the discrete logarithm in this memo.

The security levels of pairing-friendly curves are estimated by the computational cost of the most efficient algorithm for solving the above discrete logarithm problems. The best-known algorithms for solving the discrete logarithm problems are based on Pollard's rho algorithm [Pollard78] and Index Calculus [HR83]. To make index calculus algorithms more efficient, number field sieve (NFS) algorithms are utilized.

3.2. Impact of Recent Attacks

In 2016, Kim and Barbulescu proposed a new variant of the NFS algorithms, the extended tower number field sieve (exTNFS), which drastically reduces the complexity of solving FFDLP [KB16]. The exTNFS improves the polynomial selection that is the first step in the number field sieve algorithm for discrete logarithms in GF(p^k). The idea is applicable when the embedding degree k is a composite that satisfies k = i * j (gcd (i, j) = 1, i, j> 1). Since the above condition is satisfied especially when k = 2^n*3^m (n, m> 1), BN curves and BLS curves whose embedding degree is divisible by 6 are affected by the exTNFS. The basic idea of the exTNFS is based on the equality GF(p^k) = (GF(p^i)^j) and one of the improvement for reducing the amount of cost for solving FFDLP is using sub-field calculation. Please refer to [KB16] for detailed ideas and calculation algorithms of exTNFS. Due to exTNFS, the security levels of certain pairing-friendly curves asymptotically dropped down. For instance, Barbulescu and Duquesne estimated that the security of the BN curves, which had been believed to provide 128-bit security (BN256, for example) was reduced to approximately 100 bits [BD18]. Here, the security levels described in this memo correspond to the security strength of NIST recommendation [NIST].

There has since been research into the minimum bit length of the parameters of pairing-friendly curves for each security level when applying exTNFS as an attacking method for FFDLP. For 128-bit security, Barbulescu and Duquesne estimated the minimum bit length of p of BN curves and BLS12 curves after exTNFS as 461 bits [BD18]. For 256-bit security, Kiyomura et al. estimated the minimum bit length of p^k of BLS48 curves as 27,410 bits, which indicated 572 bits of p [KIK17].

4. Selection of Pairing-Friendly Curves

In this section, we introduce some of the known secure pairing-friendly curves that consider the impact of exTNFS.

First, we show the adoption status of pairing-friendly curves in standards, libraries and applications, and classify them in accordance with the 128-bit, 192-bit, and 256-bit security levels. Then, from the viewpoints of "security" and "widely used", pairing-friendly curves corresponding to each security level are selected and their parameters are indicated.

In our selection policy, it is important that selected curves are shown in peer-reviewed papers for security and that they are widely used in cryptographic libraries. In addition, "efficiency" is one of the important aspects but greatly dependant on implementations, so we choose to prioritize "security" and "widely used" over "efficiency" in consideration of future interconnections and interoperability over the internet.

As a result, we recommend the BLS curve with 381-bit characteristic of embedding degree 12 and the BN curve with the 462-bit characteristic for the 128-bit security level, and the BLS curves of embedding degree 48 with the 581-bit characteristic for the 256-bit security level. On the other hand, we do not show the parameters for 192-bit security here because there are no curves that match our selection policy.

4.1. Adoption Status of Pairing-friendly Curves

We show the pairing-friendly curves that have been selected by existing standards, cryptographic libraries, and applications.

Table 1 summarizes the adoption status of pairing-friendly curves. In this table, "Arnd" is an abbreviation for "Around". The curves categorized as 'Arnd 128-bit', 'Arnd 192-bit' and 'Arnd 256-bit' for each label show that their security levels are within the range of plus/minus 5 bits for each security level. Other labels shown with '~' mean that the security level of the categorized curve is outside the range of each security level. Specifically, the security level of the categorized curves is more than the previous column and is less than the next column. The details are described as the following subsections. A BN curve with a XXX-bit characteristic p is denoted as BNXXX and a BLS curve of embedding degree k with a XXX-bit p is denoted as BLSk_XXX.

Table 1 omits parameters with security levels below the "Arnd 128-bit" range due to space limitations and viewpoints of secure usage of parameters. On the other hand, indicating which standards, libraries, and applications use these lower security level parameters would be useful information for implementers, therefore Appendix D shows these parameters. In addition, the full version of Table 1 is available at https://lepidum.co.jp/blog/2020-03-27/ietf-draft-pfc/.

In Table 1, the security level for each curve is evaluated in accordance with [BD18],[GMT19], [MAF19] and [FK18]. Note that the Freeman curves and MNT curves are not included in this table because [BD18] does not show the security levels of these curves.

Category	Name	Curve Type	Security Levels (bit)				
			Arnd 128	~	Arnd 192	~	Arnd 256
Standard	ISO/IEC	BN384	X				
		BN512I		X			
	TCG	BN638		X			
	FIDO/W3C	BN512I		X			
		BN638		X			
Library	mcl	BLS12_381	X				
		BN382M	X				
		BN462	X				
	RELIC	BLS12_381	X				
		BLS12_446	X				
		BLS12_455	X				
		BLS12_638		X			
		BLS24_477			X		
		BLS48_575					X
		BN382R	X				
		BN446	X				
		BN638		X			
		CP8_544	X				
		K54_569					X
		KSS18_508		X			
		OT8_511	X				

	AMCL	BLS12_381	X				
		BLS12_383	X				
		BLS12_461	X				
		BLS24_479			X		
		BLS48_556					X
		BN512I		X			
	Kyushu Univ.	BLS48_581					X
	MIRACL	BLS12_381	X				
		BLS12_383	X				
		BLS12_461	X				
		BLS24_479			X		
		BLS48_556					X
		BLS48_581					X
		BN462	X				
		BN512I		X			
	Adjoint	BLS12_381	X				
		BN462	X				
	bls12377js	BLS12_377	X				
Application	Zcash	BLS12_381	X				
	Ethereum	BLS12_381	X				
	Chia Network	BLS12_381	X				
	DFINITY	BLS12_381	X				
		BN382M	X				

		BN462	X				
	Algorand	BLS12_381	X				

Table 1: Adoption Status of Pairing-Friendly Curves

4.1.1. International Standards

ISO/IEC 15946 series specifies public-key cryptographic techniques based on elliptic curves. ISO/IEC 15946-5 [ISOIEC15946-5] shows numerical examples of MNT curves[MNT01] with 160-bit p and 256-bit p, Freeman curves [Freeman06] with 224-bit p and 256-bit p, and BN curves with 160-bit p, 192-bit p, 224-bit p, 256-bit p, 384-bit p, and 512-bit p. These parameters do not take into account the effects of the exTNFS. On the other hand, the parameters may be revised in future versions since ISO/IEC 15946-5 is currently under development. As described below, BN curves with 256-bit p and 512-bit p specified in ISO/IEC 15946-5 used by other standards and libraries, these curves are especially denoted as BN256I and BN512I. The suffix 'I' of BN256I and BN512I are given from the initials of the standard name ISO.

TCG adopts the BN256I and a BN curve with 638-bit p specified by their own[TPM]. FIDO Alliance [FIDO] and W3C [W3C] adopt BN256I, BN512I, the BN638 by TCG, and the BN curve with 256-bit p proposed by Devegili et al.[DSD07] (named BN256D). The suffix 'D' of BN256D is given from the initials of the first author's name of the paper which proposed the parameter.

4.1.2. Cryptographic Libraries

There are a lot of cryptographic libraries that support pairing calculations.

PBC is a library for pairing-based cryptography published by Stanford University that supports BN curves, MNT curves, Freeman curves, and supersingular curves [PBC]. Users can generate pairing parameters by using PBC and use pairing operations with the generated parameters.

mcl[mcl] is a library for pairing-based cryptography that supports four BN curves and BLS12_381 [GMT19]. These BN curves include BN254 proposed by Nogami et al. [NASKM08] (named BN254N), BN_SNARK1 suitable for SNARK applications[libsnark], BN382M, and BN462. The suffix 'N' of BN256N and the suffix 'M' of BN382M are respectively given from the initials of the first author's name of the proposed paper and the library's name mcl. Kyushu University published a library that supports the BLS48_581 [BLS48]. The University of

Tsukuba Elliptic Curve and Pairing Library (TEPLA) [TEPLA] supports two BN curves, BN254N and BN254 proposed by Beuchat et al. [BGMORT10] (named BN254B). The suffix 'B' of BN254B is given from the initials of the first author's name of the proposed paper. Intel published a cryptographic library named Intel Integrated Performance Primitives (Intel-IPP) [Intel-IPP] and the library supports BN256I.

RELIC [RELIC] uses various types of pairing-friendly curves including six BN curves (BN158, BN254R, BN256R, BN382R, BN446, and BN638), where BN254R, BN256R, and BN382R are RELIC specific parameters that are different from BN254N, BN254B, BN256I, BN256D, and BN382M. The suffix 'R' of BN382R is given from the initials of the library's name RELIC. In addition, RELIC supports six BLS curves (BLS12_381, BLS12_446, BLS12_445, BLS12_638, BLS24_477, and BLS48_575 [MAF19]), Cocks-Pinch curves of embedding degree 8 with 544-bit p[GMT19], pairing-friendly curves constructed by Scott et al. [SG19] based on Kachisa-Scott-Schaefer curves with embedding degree 54 with 569-bit p (named K54_569)[MAF19], a KSS curve [KSS08] of embedding degree 18 with 508-bit p (named KSS18_508) [AFKMR12], Optimal TNFS-secure curve [FM19] of embedding degree 8 with 511-bit p(OT8_511), and a supersingular curve [S86] with 1536-bit p (SS_1536).

Apache Milagro Crypto Library (AMCL)[AMCL] supports four BLS curves (BLS12_381, BLS12_461, BLS24_479 and BLS48_556) and four BN curves (BN254N, BN254CX proposed by CertiVox, BN256I, and BN512I). In addition to AMCL's supported curves, MIRACL [MIRACL] supports BN462 and BLS48_581.

Adjoint published a library that supports the BLS12_381 and six BN curves (BN_SNARK1, BN254B, BN254N, BN254S1, BN254S2, and BN462) [AdjointLib], where BN254S1 and BN254S2 are BN curves adopted by an old version of AMCL [AMCLv2]. The suffix 'S' of BN254S1 and BN254S2 are given from the initials of developper's name because he proposed these parameters.

The Celo foundation published the bls12377js library [bls12377js]. The supported curve is the BLS12_377 curve which is shown in [BCGMMW20].

4.1.3. Applications

Zcash uses a BN curve (named BN128) in their library libsnark [libsnark]. In response to the exTNFS attacks, they proposed new parameters using BLS12_381 [BLS12_381] [GMT19]and published its experimental implementation [zkcrypto].

Ethereum 2.0 adopted BLS12_381 and uses the implementation by Meyer [pureGo-bls]. Chia Network published their implementation [Chia] by integrating the RELIC toolkit [RELIC]. DFINITY uses mcl, and Algorand published an implementation which supports BLS12_381.

4.2. For 128-bit Security

Table 1 shows a lot of cases of adopting BN and BLS curves. Among them, BLS12_381 and BN462 match our selection policy. Especially, the one that best matches the policy is BLS12_381 from the viewpoint of "widely used" and "efficiency", so we introduce the parameters of BLS12_381 in this memo.

On the other hand, from the viewpoint of the future use, the parameter of BN462 is also introduced. As shown in recent security evaluations for BLS12_381[BD18] [GMT19], its security level close to 128-bit but it is less than 128-bit. If the attack is improved even a little, BLS12_381 will not be suitable for the curve of the 128-bit security level. As curves of 128-bit security level are currently the most widely used, we recommend both BLS12_381 and BN462 in this memo in order to have a more efficient and a more prudent option respectively.

4.2.1. BLS Curves for the 128-bit security level (BLS12_381)

In this part, we introduce the parameters of the Barreto-Lynn-Scott curve of embedding degree 12 with 381-bit p that is adopted by a lot of applications such as Zcash [Zcash], Ethereum [Ethereum], and so on.

The BLS12_381 curve is shown in [BLS12_381] and it is defined by the parameter

$$t = -2^{63} - 2^{62} - 2^{60} - 2^{57} - 2^{48} - 2^{16}$$

where the size of p becomes 381-bit length.

For the finite field GF(p), the towers of extension field GF(p^2), GF(p^6) and GF(p^12) are defined by indeterminates u, v, and w as follows:

$$GF(p^2) = GF(p)[u] / (u^2 + 1)$$
$$GF(p^6) = GF(p^2)[v] / (v^3 - u - 1)$$
$$GF(p^{12}) = GF(p^6)[w] / (w^2 - v).$$

Defined by t, the elliptic curve E and its twist E' are represented by E: $y^2 = x^3 + 4$ and E': $y^2 = x^3 + 4(u + 1)$. BLS12_381 is categorized as M-type.

We have to note that the security level of this pairing is expected to be 126 rather than 128 bits [GMT19].

Parameters of BLS12_381 are given as follows.

* G_1 is the largest prime-order subgroup of E(GF(p))

  - BP = (x,y) : a 'base point', i.e., a generator of G_1

* G_2 is an r-order subgroup of E'(GF(p^2))

  - BP' = (x',y') : a 'base point', i.e., a generator of G_2 (encoded with [I-D.ietf-lwig-curve-representations])

    o x' = x'_0 + x'_1 * u (x'_0, x'_1 in GF(p))

    o y' = y'_0 + y'_1 * u (y'_0, y'_1 in GF(p))

  - h' : the cofactor #E'(GF(p^2))/r

p:
0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f624
1eabfffeb153ffffb9feffffffffaaab

r:
0x73eda753299d7d483339d80809a1d80553bda402fffe5bfeffffffff00000001

x:
0x17f1d3a73197d7942695638c4fa9ac0fc3688c4f9774b905a14e3a3f171bac58
6c55e83ff97a1aeffb3af00adb22c6bb

y:
0x08b3f481e3aaa0f1a09e30ed741d8ae4fcf5e095d5d00af600db18cb2c04b3ed
d03cc744a2888ae40caa232946c5e7e1

h: 0x396c8c005555e1568c00aaab0000aaab

b: 4

x'_0:
0x024aa2b2f08f0a91260805272dc51051c6e47ad4fa403b02b4510b647ae3d177
0bac0326a805bbefd48056c8c121bdb8

x'_1:
0x13e02b6052719f607dacd3a088274f65596bd0d09920b61ab5da61bbdc7f5049
334cf11213945d57e5ac7d055d042b7e

y'_0:
0x0ce5d527727d6e118cc9cdc6da2e351aadfd9baa8cbdd3a76d429a695160d12c
923ac9cc3baca289e193548608b82801

y'_1:
0x0606c4a02ea734cc32acd2b02bc28b99cb3e287e85a763af267492ab572e99ab
3f370d275cec1da1aaa9075ff05f79be

h':
0x5d543a95414e7f1091d50792876a202cd91de4547085abaa68a205b2e5a7ddfa
628f1cb4d9e82ef21537e293a6691ae1616ec6e786f0c70cf1c38e31c7238e5

b': 4 * (u + 1)

As mentioned above, BLS12_381 is adopted in a lot of applications. Since it is expected that BLS12_381 will continue to be widely used more and more in the future, Appendix C shows the serialization format of points on an elliptic curve as useful information. This serialization format is also adopted in [I-D.boneh-bls-signature] [zkcrypto].

In addition, many pairing-based cryptographic applications use a hashing to an elliptic curve procedure that outputs a rational point on an elliptic curve from an arbitrary input. A standard specification of ciphersuites for a hashing to an elliptic curve, including BLS12_381, is under discussion in the IETF [I-D.irtf-cfrg-hash-to-curve] and it will be valuable information for implementers.

4.2.2. BN Curves for the 128-bit security level (BN462)

A BN curve with the 128-bit security level is shown in [BD18], which we call BN462. BN462 is defined by the parameter

t = 2^114 + 2^101 - 2^14 - 1

for the definition in Section 2.3.

For the finite field GF(p), the towers of extension field GF(p^2), GF(p^6) and GF(p^12) are defined by indeterminates u, v, and w as follows:

```
GF(p^2) = GF(p)[u] / (u^2 + 1)
GF(p^6) = GF(p^2)[v] / (v^3 - u - 2)
GF(p^12) = GF(p^6)[w] / (w^2 - v).
```

Defined by t, the elliptic curve E and its twist E' are represented by E: y^2 = x^3 + 5 and E': y^2 = x^3 - u + 2, respectively. The size of p becomes 462-bit length. BN462 is categorized as D-type.

We have to note that BN462 is significantly slower than BLS12_381, but has 134-bit security level [GMT19], so may be more resistant to future small improvements to the exTNFS attack.

We note also that CP8_544 is about 20% faster that BN462 [GMT19], has 131-bit security level, and that due to its construction will not be affected by future small improvements to the exTNFS attack. However, as this curve is not widely used (it is only implemented in one library), we instead chose BN462 for our 'safe' option.

We give the following parameters for BN462.

* G_1 is the largest prime-order subgroup of E(GF(p))

  - BP = (x,y) : a 'base point', i.e., a generator of G_1

* G_2 is an r-order subgroup of E'(GF(p^2))

  - BP' = (x',y') : a 'base point', i.e., a generator of G_2 (encoded with [I-D.ietf-lwig-curve-representations])

    o x' = x'_0 + x'_1 * u (x'_0, x'_1 in GF(p))

    o y' = y'_0 + y'_1 * u (y'_0, y'_1 in GF(p))

  - h' : the cofactor #E'(GF(p^2))/r

p:
0x2404803601200 23ffffffffff6ff0cf6b7d9bfca0000000000d812908f41c802
0ffffffffff6ff66fc6ff687f640000000002401b00840138013

r:
0x240480360120023ffffffffff6ff0cf6b7d9bfca0000000000d812908ee1c201
f7fffffffff6ff66fc7bf717f7c0000000002401b007e010800d

x:
0x21a6d67ef250191fadba34a0a30160b9ac9264b6f95f63b3edbec3cf4b2e689d
b1bbb4e69a416a0b1e79239c0372e5cd70113c98d91f36b6980d

y:
0x0118ea0460f7f7abb82b33676a7432a490eeda842cccfa7d788c659650426e6a
f77df11b8ae40eb80f475432c66600622ecaa8a5734d36fb03de

h: 1

b: 5

x'_0:
0x0257ccc85b58dda0dfb38e3a8cbdc5482e0337e7c1cd96ed61c913820408208f
9ad2699bad92e0032ae1f0aa6a8b48807695468e3d934ae1e4df

x'_1:
0x1d2e4343e8599102af8edca849566ba3c98e2a354730cbed9176884058b18134
dd86bae555b783718f50af8b59bf7e850e9b73108ba6aa8cd283

y'_0:
0x0a0650439da22c1979517427a20809eca035634706e23c3fa7a6bb42fe810f13
99a1f41c9ddae32e03695a140e7b11d7c3376e5b68df0db7154e

y'_1:
0x073ef0cbd438cbe0172c8ae37306324d44d5e6b0c69ac57b393f1ab370fd725c
c647692444a04ef87387aa68d53743493b9eba14cc552ca2a93a

h':
0x240480360120023ffffffffff6ff0cf6b7d9bfca0000000000d812908fa1ce02
27fffffffff6ff66fc63f5f7f4c0000000002401b008a0168019

b': -u + 2

4.3. For 256-bit Security

As shown in Table 1, there are three candidates of pairing-friendly curves for 256-bit security. According to our selection policy, we select BLS48_581, as it is the most widely adopted by cryptographic libraries.

The selected BLS48 curve is shown in [KIK17] and it is defined by the parameter

t = -1 + 2^7 - 2^10 - 2^30 - 2^32.

In this case, the size of p becomes 581-bit.

For the finite field GF(p), the towers of extension field GF(p^2), GF(p^4), GF(p^8), GF(p^24) and GF(p^48) are defined by indeterminates u, v, w, z, and s as follows:

```
GF(p^2) = GF(p)[u] / (u^2 + 1)
GF(p^4) = GF(p^2)[v] / (v^2 + u + 1)
GF(p^8) = GF(p^4)[w] / (w^2 + v)
GF(p^24) = GF(p^8)[z] / (z^3 + w)
GF(p^48)= GF(p^24)[s] / (s^2 + z).
```

The elliptic curve E and its twist E' are represented by E: y^2 = x^3 + 1 and E': y^2 = x^3 - 1 / w. BLS48_581 is categorized as D-type.

We then give the parameters for BLS48_581 as follows.

* G_1 is the largest prime-order subgroup of E(GF(p))

  - BP = (x,y) : a 'base point', i.e., a generator of G_1

* G_2 is an r-order subgroup of E'(GF(p^8))

  - BP' = (x',y') : a 'base point', i.e., a generator of G_2 (encoded with [I-D.ietf-lwig-curve-representations])

    o x' = x'_0 + x'_1 * u + x'_2 * v + x'_3 * u * v + x'_4 * w + x'_5 * u * w + x'_6 * v * w + x'_7 * u * v * w (x'_0, ..., x'_7 in GF(p))

    o y' = y'_0 + y'_1 * u + y'_2 * v + y'_3 * u * v + y'_4 * w + y'_5 * u * w + y'_6 * v * w + y'_7 * u * v * w (y'_0, ..., y'_7 in GF(p))

  - h' : the cofactor #E'(GF(p^8))/r

p:
0x1280f73ff3476f313824e31d47012a0056e84f8d122131bb3be6c0f1f3975444
a48ae43af6e082acd9cd30394f4736daf68367a5513170ee0a578fdf721a4a48ac
3edc154e6565912b

r:
0x2386f8a925e2885e233a9ccc1615c0d6c635387a3f0b3cbe003fad6bc972c2e6
e741969d34c4c92016a85c7cd0562303c4ccbe599467c24da118a5fe6fcd671c01

x:
0x02af59b7ac340f2baf2b73df1e93f860de3f257e0e86868cf61abdbaedffb9f7
544550546a9df6f9645847665d859236ebdbc57db368b11786cb74da5d3a1e6d8c
3bce8732315af640

y:
0x0cefda44f6531f91f86b3a2d1fb398a488a553c9efeb8a52e991279dd41b720e
f7bb7beffb98aee53e80f678584c3ef22f487f77c2876d1b2e35f37aef7b926b57
6dbb5de3e2587a70

x'_0:
0x05d615d9a7871e4a38237fa45a2775debabbefc70344dbccb7de64db3a2ef156
c46ff79baad1a8c42281a63ca0612f400503004d80491f510317b79766322154de
c34fd0b4ace8bfab

x'_1:
0x07c4973ece2258512069b0e86abc07e8b22bb6d980e1623e9526f6da12307f4e
1c3943a00abfedf16214a76affa62504f0c3c7630d979630ffd75556a01afa143f
1669b36676b47c57

x'_2:
0x01fccc70198f1334e1b2ea1853ad83bc73a8a6ca9ae237ca7a6d6957ccbab5ab
6860161c1dbd19242ffae766f0d2a6d55f028cbdfbb879d5fea8ef4cded6b3f0b4
6488156ca55a3e6a

x'_3:
0x0be2218c25ceb6185c78d8012954d4bfe8f5985ac62f3e5821b7b92a393f8be0
cc218a95f63e1c776e6ec143b1b279b9468c31c5257c200ca52310b8cb4e80bc3f
09a7033cbb7feafe

x'_4:
0x038b91c600b35913a3c598e4caa9dd63007c675d0b1642b5675ff0e7c5805386
699981f9e48199d5ac10b2ef492ae589274fad55fc1889aa80c65b5f746c9d4cbb
739c3a1c53f8cce5

x'_5:
0x0c96c7797eb0738603f1311e4ecda088f7b8f35dcef0977a3d1a58677bb03741
8181df63835d28997eb57b40b9c0b15dd7595a9f177612f097fc7960910fce3370
f2004d914a3c093a

x'_6:
0x0b9b7951c6061ee3f0197a498908aee660dea41b39d13852b6db908ba2c0b7a4
49cef11f293b13ced0fd0caa5efcf3432aad1cbe4324c22d63334b5b0e205c3354
e41607e60750e057

x'_7:
0x0827d5c22fb2bdec5282624c4f4aaa2b1e5d7a9defaf47b5211cf741719728a7
f9f8cfca93f29cff364a7190b7e2b0d4585479bd6aebf9fc44e56af2fc9e97c3f8
4e19da00fbc6ae34

y'_0:
0x00eb53356c375b5dfa497216452f3024b918b4238059a577e6f3b39ebfc435fa
ab0906235afa27748d90f7336d8ae5163c1599abf77eea6d659045012ab12c0ff3
23edd3fe4d2d7971

y'_1:
0x0284dc75979e0ff144da6531815fcadc2b75a422ba325e6fba01d72964732fcb
f3afb096b243b1f192c5c3d1892ab24e1dd212fa097d760e2e588b423525ffc7b1
11471db936cd5665

```
y'_2:
   0x0b36a201dd008523e421efb70367669ef2c2fc5030216d5b119d3a480d370514
   475f7d5c99d0e90411515536ca3295e5e2f0c1d35d51a652269cbc7c46fc3b8fde
   68332a526a2a8474

y'_3:
   0x0aec25a4621edc0688223fbbd478762b1c2cded3360dcee23dd8b0e710e122d2
   742c89b224333fa40dced2817742770ba10d67bda503ee5e578fb3d8b8a1e53373
   16213da92841589d

y'_4:
   0x0d209d5a223a9c46916503fa5a88325a2554dc541b43dd93b5a959805f112985
   7ed85c77fa238cdce8a1e2ca4e512b64f59f430135945d137b08857fdddfcf7a43
   f47831f982e50137

y'_5:
   0x07d0d03745736b7a513d339d5ad537b90421ad66eb16722b589d82e2055ab750
   4fa83420e8c270841f6824f47c180d139e3aafc198caa72b679da59ed8226cf3a5
   94eedc58cf90bee4

y'_6:
   0x0896767811be65ea25c2d05dfdd17af8a006f364fc0841b064155f14e4c819a6
   df98f425ae3a2864f22c1fab8c74b2618b5bb40fa639f53dccc9e884017d9aa62b
   3d41faeafeb23986

y'_7:
   0x035e2524ff89029d393a5c07e84f981b5e068f1406be8e50c87549b6ef8eca9a
   9533a3f8e69c31e97e1ad0333ec719205417300d8c4ab33f748e5ac66e84069c55
   d667ffcb732718b6

h:  0x85555841aaaec4ac

b:  1

h':
   0x170e915cb0a6b7406b8d94042317f811d6bc3fc6e211ada42e58ccfcb3ac076a
   7e4499d700a0c23dc4b0c078f92def8c87b7fe63e1eea270db353a4ef4d38b5998
   ad8f0d042ea24c8f02be1c0c83992fe5d7725227bb27123a949e0876c0a8ce0a67
   326db0e955dcb791b867f31d6bfa62fbdd5f44a00504df04e186fae033f1eb43c1
   b1a08b6e086eff03c8fee9ebdd1e191a8a4b0466c90b389987de5637d5dd13dab3
   3196bd2e5afa6cd19cf0fc3fc7db7ece1f3fac742626b1b02fcee04043b2ea9649
   2f6afa51739597c54bb78aa6b0b99319fef9d09f768831018ee6564c68d054c62f
   2e0b4549426fec24ab26957a669dba2a2b6945ce40c9aec6afdeda16c79e15546c
   d7771fa544d5364236690ea06832679562a68731420ae52d0d35a90b8d10b688e3
   1b6aee45f45b7a5083c71732105852decc888f64839a4de33b99521f0984a418d2
   0fc7b0609530e454f0696fa2a8075ac01cc8ae3869e8d0fe1f3788ffac4c01aa27
   20e431da333c83d9663bfb1fb7a1a7b90528482c6be7892299030bb51a51dc7e91
   e9156874416bf4c26f1ea7ec578058563960ef92bbbb8632d3a1b695f954af10e9
```

a78e40acffc13b06540aae9da5287fc4429485d44e6289d8c0d6a3eb2ece350124
52751839fb48bc14b515478e2ff412d930ac20307561f3a5c998e6bcbfebd97eff
c6433033a2361bfcdc4fc74ad379a16c6dea49c209b1

b’: -1 / w

5. Security Considerations

The recommended pairing-friendly curves are selected by considering the exTNFS proposed by Kim et al. in 2016 [KB16] and they are categorized in each security level in accordance with [BD18]. Implementers who will newly develop pairing-based cryptography applications SHOULD use the recommended parameters. As of 2020, as far as we’ve investigated the top cryptographic conferences in the past, there are no fatal attacks that significantly reduce the security of pairing-friendly curves after exTNFS.

BLS curves of embedding degree 12 typically require a characteristic p of 461 bits or larger to achieve the 128-bit security level [BD18]. Note that the security level of BLS12_381, which is adopted by a lot of libraries and applications, is slightly below 128 bits because a 381-bit characteristic is used [BD18] [GMT19].

BN254 is used in most of the existing implementations as shown in Section 4.1 (and Appendix D), however, BN curves that were estimated as the 128-bit security level before exTNFS including BN254 ensure no more than the 100-bit security level by the effect of exTNFS.

In addition, implementors should be aware of the following points when they implement pairing-based cryptographic applications using recommended curves. Regarding the use case and applications of pairing-based cryptographic applications, please refer Section 1.2.

In applications such as key agreement protocols, users exchange the elements in G_1 and G_2 as public keys. To check these elements are so-called sub-group secure [BCM15], implementors should validate if the elements have the correct order r. Specifically, for public keys P in G_1 and Q in G_2, a receiver should calculate scalar multiplications [r]P and [r]Q, and check the results become points at infinity.

The pairing-based protocols, such as the BLS signatures, use a scalar multiplication in G_1, G_2 and an exponentiation in G_3 with the secret key. In order to prevent the leakage of secret key due to side channel attacks, implementors should apply countermeasure techniques such as montgomery ladder [Montgomery] [CF06] when they implement modules of a scalar multiplication and an exponentiation. Please refer [Montgomery] and [CF06] for the detailed algorithms of montgomery ladder.

When converting between an element in extension field and an octet string, implementors should check that the coefficient is within an appropriate range [IEEE1363]. If the coefficient is out of range, there is a possible that security vulnerabilities such as the signature forgery may occur.

Recommended parameters are affected by the Cheon’s attack which is a solving algorithm for the strong DH problem [Cheon06]. The mathematical problem that provides the security of the strong DH problem is called ECDLP with Auxiliary Inputs (ECDLPwAI). In ECDLPwAI, given rational points P, [K]P, [K^i]P, for i=1,...,n, then we find a secret K. Since the complexity of ECDLPwAI is given as O(sqrt((r-1)/n + sqrt(n)) where n|r-1 by using Cheon’s algorithm whereas the complexity of ECDLP is given as O(sqrt(r)), the complexity of ECDLPwAI with the ideal value n becomes dramatically smaller than that of ECDLP. Please refer [Cheon06] for the details of Cheon’s algorithm. Therefore, implementers should be careful when they design cryptographic protocols based on the strong DH problem. For example, in the case of Short Signatures, they can prevent the Cheon’s attack by carefully setting the maximum number of queries which corresponds to the parameter n.

6. IANA Considerations

This document has no actions for IANA.

7. Acknowledgements

The authors would like to appreciate a lot of authors including Akihiro Kato for their significant contribution to early versions of this memo. The authors would also like to acknowledge Kim Taechan, Hoeteck Wee, Sergey Gorbunov, Michael Scott, Chloe Martindale as an Expert Reviewer, Watson Ladd, Armando Faz, Rene Struik, and Satoru Kanno for their valuable comments.

8. References

8.1. Normative References

[BD18] Barbulescu, R. and S. Duquesne, "Updating Key Size Estimations for Pairings", DOI 10.1007/s00145-018-9280-5, Journal of Cryptology, January 2018, <https://doi.org/10.1007/s00145-018-9280-5>.

[BLS02] Barreto, P., Lynn, B., and M. Scott, "Constructing Elliptic Curves with Prescribed Embedding Degrees", DOI 10.1007/3-540-36413-7_19, Security in Communication Networks pp. 257-267, 2003, <https://doi.org/10.1007/3-540-36413-7_19>.

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

[GMT19] Guillevic, A., Masson, S., and E. Thome, "Cocks窶撤inch curves of embedding degrees five to eight and optimal ate pairing computation", DOI 10.1007/s10623-020-00727-w, International Journal of Designs, Codes and Cryptography vol. 88, pp. 1047-1081, 2019, <https://doi.org/10.1007/s10623-020-00727-w>.

[KB16] Kim, T. and R. Barbulescu, "Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case", DOI 10.1007/978-3-662-53018-4_20, Advances in Cryptology - CRYPTO 2016 pp. 543-571, 2016, <https://doi.org/10.1007/978-3-662-53018-4_20>.

[KIK17] Kiyomura, Y., Inoue, A., Kawahara, Y., Yasuda, M., Takagi, T., and T. Kobayashi, "Secure and Efficient Pairing at 256-Bit Security Level", DOI 10.1007/978-3-319-61204-1_4, Applied Cryptography and Network Security pp. 59-79, 2017, <https://doi.org/10.1007/978-3-319-61204-1_4>.

[NIST] Barker, E., "NIST special publication 800-57 part 1 (revised) : Recommendation for key management, part 1: General (revised)", National Institute of Standards and Technology (NIST), 2020.

[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>.

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

[Ver09] Vercauteren, F., "Optimal Pairings", DOI 10.1109/tit.2009.2034881, IEEE Transactions on Information Theory Vol. 56, pp. 455-461, January 2010, <https://doi.org/10.1109/tit.2009.2034881>.

8.2. Informative References

[AdjointLib]
Adjoint Inc., "Optimised bilinear pairings over elliptic curves", 2018, <https://github.com/adjoint-io/pairing>.

[AFKMR12] Aranha, D.F., Fuentes-Castaneda, L., Knapp, E., Menezes, A., and F. Rodrテュguez-Henrテュquez, "Implementing Pairings at the 192-Bit Security Level", DOI /10.1007/978-3-642-36334-4_11, Pairing 2012 pp. 177-195, 2012, <https://doi.org//10.1007/978-3-642-36334-4_11>.

[Algorand] Gorbunov, S., "Efficient and Secure Digital Signatures for Proof-of-Stake Blockchains", <https://medium.com/algorand/digital-signatures-for-blockchains-5820e15fbe95>.

[AMCL] The Apache Software Foundation, "The Apache Milagro Cryptographic Library (AMCL)", 2016, <https://github.com/apache/incubator-milagro-crypto>.

[AMCLv2] The Apache Software Foundation, "Old version of the Apache Milagro Cryptographic Library", 2016, <https://github.com/miracl/amcl/tree/master/version22>.

[BCGMMW20] Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., and H. Wu, "ZEXE: Enabling Decentralized Private Computation", DOI 10.1109/SP40000.2020.00050, IEEE Symposium on Security and Privacy 2020, 2020, <https://doi.org/10.1109/SP40000.2020.00050>.

[BCM15] Barreto, P. S. L. M., Costello, C., Misoczki, R., Naehrig, M., Pereira, G. C. C. F., and G. Zanon, "Subgroup security in pairing-based cryptography", Cryptology ePrint Archive Report 2015/247, 2015, <https://eprint.iacr.org/2015/247.pdf>.

[BGMORT10] Beuchat, J., Gonzテ。lez-Dテュaz, J., Mitsunari, S., Okamoto, E., Rodrテュguez-Henrテュquez, F., and T. Teruya, "High-Speed Software Implementation of the Optimal Ate Pairing over Barreto褰哲aehrig Curves", DOI 10.1007/978-3-642-17455-1_2, Pairing 2010 pp. 21-39, 2010, <https://doi.org/10.1007/978-3-642-17455-1_2>.

[BL10] Brickell, E. and J. Li, "Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation", DOI 10.1109/socialcom.2010.118, 2010 IEEE Second International Conference on Social Computing, August 2010, <https://doi.org/10.1109/socialcom.2010.118>.

[bls12377js] The Celo Foundation, "bls12377js", 2019, <https://github.com/celo-org/bls12377js>.

[BLS12_381] Bowe, S., "BLS12-381: New zk-SNARK Elliptic Curve Construction", <https://electriccoin.co/blog/new-snark-curve/>.

[BLS48] Kyushu University, "bls48 - C++ library for Optimal Ate Pairing on BLS48", 2017, <https://github.com/mk-math-kyushu/bls48>.

[CCS07] Chen, L., Cheng, Z., and N. Smart, "Identity-based key agreement protocols from pairings", DOI 10.1007/s10207-006-0011-9, International Journal of Information Security Vol. 6, pp. 213-241, January 2007, <https://doi.org/10.1007/s10207-006-0011-9>.

[CF06] Cohen, H. and G. Frey, "Handbook of Elliptic and Hyperelliptic Curve Cryptography", DOI 10.1201/9780367801625, Chapman and Hall CRC, 2006, <https://doi.org/10.1201/9780367801625>.

[Cheon06] Cheon, J. H., "Security Analysis of the Strong Diffie-Hellman Problem", DOI 10.1007/11761679_1, EUROCRYPT 2006 pp. 1-11, 2006, <https://doi.org/10.1007/11761679_1>.

[Chia] Chia Network, "BLS signatures in C++, using the relic toolkit", <https://github.com/Chia-Network/bls-signatures>.

[CIRCL] Cloudflare, "CIRCL: Cloudflare Interoperable, Reusable Cryptographic Library", 2019, <https://github.com/cloudflare/circl>.

[CLN09] Costello, C., Lange, T., and M. Naehrig, "Faster Pairing Computations on Curves with High-Degree Twists", Cryptology ePrint Archive Report 2009/615, 2009, <https://eprint.iacr.org/2009/615.pdf>.

[Cloudflare]
Sullivan, N., "Geo Key Manager: How It Works", <https://blog.cloudflare.com/geo-key-manager-how-it-works/>.

[DFINITY] Williams, D., "DFINITY Technology Overview Series Consensus System Rev. 1", n.d., <https://dfinity.org/pdf-viewer/library/dfinity-consensus.pdf>.

[DSD07] Devegili, A. J., Scott, M., and R. Dahab, "Implementing Cryptographic Pairings over Barreto-Naehrig Curves", DOI 10.1007/978-3-540-73489-5_10, Pairing 2007 pp. 197-207, 2007, <https://doi.org/10.1007/978-3-540-73489-5_10>.

[ECRYPT] ECRYPT, "Final Report on Main Computational Assumptions in Cryptography".

[EPID] Intel Corporation, "Intel (R) SGX: Intel (R) EPID Provisioning and Attestation Services", <https://software.intel.com/en-us/download/intel-sgx-intel-epid-provisioning-and-attestation-services>.

[Ethereum] Jordan, R., "Ethereum 2.0 Development Update #17 - Prysmatic Labs", <https://medium.com/prysmatic-labs/ethereum-2-0-development-update-17-prysmatic-labs-ed5bcf82ec00>.

[FIDO] Lindemann, R., "FIDO ECDAA Algorithm - FIDO Alliance Review Draft 02", <https://fidoalliance.org/specs/fido-v2.0-rd-20180702/fido-ecdaa-algorithm-v2.0-rd-20180702.html>.

[FK18] Fotiadis, G. and E. Konstantinou, "TNFS Resistant Families of Pairing-Friendly Elliptic Curves", Cryptology ePrint Archive Report 2018/1017, 2018, <https://eprint.iacr.org/2018/1017.pdf>.

[FM19] Fotiadis, G. and C. Martindale, "Optimal TNFS-secure pairings on elliptic curves with composite embedding degree", Cryptology ePrint Archive Report 2019/555, 2019, <https://eprint.iacr.org/2019/555.pdf>.

[Freeman06]
Freeman, D., "Constructing pairing-friendly elliptic curves with embedding degree 10", DOI 10.1007/11792086_32, ANTS 2006 pp. 452-465, 2006, <https://doi.org/10.1007/11792086_32>.

[FSU10] Fujioka, A., Suzuki, K., and B. Ustaoglu, "Ephemeral Key Leakage Resilient and Efficient ID-AKEs That Can Share Identities, Private and Master Keys", DOI 10.1007/978-3-642-17455-1_12, Lecture Notes in Computer Science pp. 187-205, 2010, <https://doi.org/10.1007/978-3-642-17455-1_12>.

[HR83] Hellman, M. and J. Reyneri, "Fast Computation of Discrete Logarithms in GF (q)", DOI 10.1007/978-1-4757-0602-4_1, Advances in Cryptology pp. 3-13, 1983, <https://doi.org/10.1007/978-1-4757-0602-4_1>.

[I-D.boneh-bls-signature]
Boneh, D., Gorbunov, S., Wee, H., and Z. Zhang, "BLS Signature Scheme", Work in Progress, Internet-Draft, draft-boneh-bls-signature-00, 8 February 2019, <https://datatracker.ietf.org/doc/html/draft-boneh-bls-signature-00>.

[I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", Work in Progress, Internet-Draft, draft-ietf-lwig-curve-representations-08, 24 July 2019, <https://datatracker.ietf.org/doc/html/draft-ietf-lwig-curve-representations-08>.

[I-D.irtf-cfrg-hash-to-curve]
Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R. S., and C. A. Wood, "Hashing to Elliptic Curves", Work in Progress, Internet-Draft, draft-irtf-cfrg-hash-to-curve-09, 29 June 2020, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-09>.

[IEEE1363] "IEEE Standard Specifications for Public-Key Cryptography", IEEE standard, DOI 10.1109/IEEESTD.2000.92292, 2000, <https://doi.org/10.1109/IEEESTD.2000.92292>.

[Intel-IPP]
Intel Corporation, "Developer Reference for Intel Integrated Performance Primitives Cryptography 2019", 2018, <https://software.intel.com/en-us/ipp-crypto-reference-arithmetic-of-the-group-of-elliptic-curve-points>.

[ISOIEC11770-3]
ISO/IEC, "ISO/IEC 11770-3:2015", ISO/IEC Information technology -- Security techniques -- Key management -- Part 3: Mechanisms using asymmetric techniques, 2015.

[ISOIEC15946-5]
ISO/IEC, "ISO/IEC 15946-5:2017", ISO/IEC Information technology -- Security techniques -- Cryptographic techniques based on elliptic curves -- Part 5: Elliptic curve generation, 2017.

[Joux00] Joux, A., "A One Round Protocol for Tripartite Diffie-Hellman", DOI 10.1007/10722028_23, Lecture Notes in Computer Science pp. 385-393, 2000, <https://doi.org/10.1007/10722028_23>.

[KSS08] Kachisa, E., Schaefer, E., and M. Scott, "Constructing Brezing-Weng Pairing-Friendly Elliptic Curves Using Elements in the Cyclotomic Field", DOI 10.1007/978-3-540-85538-5_9, Pairing 2008 pp. 126-135, 2008, <https://doi.org/10.1007/978-3-540-85538-5_9>.

[libsnark] SCIPR Lab, "libsnark: a C++ library for zkSNARK proofs", 2012, <https://github.com/zcash/libsnark>.

[M-Pin] Scott, M., "M-Pin: A Multi-Factor Zero Knowledge Authentication Protocol", July 2019, <https://www.miracl.com/miracl-labs/m-pin-a-multi-factor-zero-knowledge-authentication-protocol>.

[MAF19] Mbiang, N.B., Aranha, D.F., and E. Fouotsa, "Computing the Optimal Ate Pairing over Elliptic Curves with Embedding Degrees 54 and 48 at the 256-bit security level", International Journal of Applied Cryptography to appear, 2019, <https://www.researchgate.net/publication/337011283_Computing_the_Optimal_Ate_Pairing_over_Elliptic_Curves_with_Embedding_Degrees_54_and_48_at_the_256-bit_security_level>.

[mcl] Mitsunari, S., "mcl - A portable and fast pairing-based cryptography library", 2016, <https://github.com/herumi/mcl>.

[MIRACL] MIRACL Ltd., "The MIRACL Core Cryptographic Library", 2019, <https://github.com/miracl/core>.

[MNT01] Miyaji, A., Nakabayashi, M., and S. Takano, "New explicit conditions of Elliptic Curve Traces under FR reduction", IEICE Trans. Fundamentals. E84-A(5) pp. 1234-1243, 2001.

[Montgomery] Montgomery, P., "Speeding the Pollard and Elliptic Curve Methods of Factorization", MATHEMATICS OF COMPUTATION , January, 1987, <https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/S0025-5718-1987-0866113-7.pdf>.

[MP04] Guillevic, A., Masson, S., and E. Thome, "Cocks寠撤inch curves of embedding degrees five to eight and optimal ate pairing computation", Cryptology ePrint Archive Report 2019/431, 2019, <https://eprint.iacr.org/2004/032.pdf>.

[NASKM08] Nogami, Y., Akane, M., Sakemi, Y., Kato, H., and Y. Morikawa, "Integer Variable X-Based Ate Pairing", DOI 10.1007/978-3-540-85538-5_13, Pairing 2008 pp. 178-191, 2008, <https://doi.org/10.1007/978-3-540-85538-5_13>.

[PBC] Lynn, B., "PBC Library - The Pairing-Based Cryptography Library", 2006, <https://crypto.stanford.edu/pbc/>.

[Pollard78] Pollard, J., "Monte Carlo methods for index computation $({\rm mod}\ p)$", DOI 10.1090/s0025-5718-1978-0491431-9, Mathematics of Computation Vol. 32, pp. 918-918, September 1978, <https://doi.org/10.1090/s0025-5718-1978-0491431-9>.

[pureGo-bls] Meyer, J., "Pure GO bls library", 2019, <https://github.com/phoreproject/bls>.

[RELIC] Gouvea, C.P.L., "RELIC is an Efficient LIbrary for Cryptography", 2013, <https://github.com/relic-toolkit/relic>.

[RFC5091] Boyen, X. and L. Martin, "Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems", RFC 5091, DOI 10.17487/RFC5091, December 2007, <https://www.rfc-editor.org/info/rfc5091>.

[RFC6508] Groves, M., "Sakai-Kasahara Key Encryption (SAKKE)", RFC 6508, DOI 10.17487/RFC6508, February 2012, <https://www.rfc-editor.org/info/rfc6508>.

[RFC6509] Groves, M., "MIKEY-SAKKE: Sakai-Kasahara Key Encryption in Multimedia Internet KEYing (MIKEY)", RFC 6509, DOI 10.17487/RFC6509, February 2012, <https://www.rfc-editor.org/info/rfc6509>.

[RFC6539] Cakulev, V., Sundaram, G., and I. Broustis, "IBAKE: Identity-Based Authenticated Key Exchange", RFC 6539, DOI 10.17487/RFC6539, March 2012, <https://www.rfc-editor.org/info/rfc6539>.

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

[S86] Silverman, J. H., "The arithmetic of elliptic curves", Springer GTM 106, 1986.

[SAKKE] 3GPP, "Security of the mission critical service (Release 15)", 3GPP TS 33.180 15.3.0, 2018.

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

[SG19] Scott, M. and A. Guillevic, "A New Family of Pairing-Friendly elliptic curves", Cryptology ePrint Archive Report 2019/193, 2019, <https://eprint.iacr.org/2018/193.pdf>.

[TEPLA] University of Tsukuba, "TEPLA: University of Tsukuba Elliptic Curve and Pairing Library", 2013, <http://www.cipher.risk.tsukuba.ac.jp/tepla/index_e.html>.

[TPM] Trusted Computing Group (TCG), "Trusted Platform Module Library Specification, Family \"2.0\", Level 00, Revision 01.38", <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

[W3C] Lundberg, E., "Web Authentication: An API for accessing Public Key Credentials Level 1 - W3C Recommendation", <https://www.w3.org/TR/webauthn/>.

[Zcash] Lindemann, R., "What are zk-SNARKs?", <https://z.cash/technology/zksnarks.html>.

[ZCashRep] Electric Coin Company, "BLS12-381", July 2017, <https://github.com/zkcrypto/pairing/blob/master/src/bls12_381/README.md>.

[zkcrypto] zkcrypto, "zkcrypto - Pairing-friendly elliptic curve library", 2017, <https://github.com/zkcrypto/pairing>.

Appendix A. Computing the Optimal Ate Pairing

Before presenting the computation of the optimal Ate pairing e(P, Q) satisfying the properties shown in Section 2.2, we give the subfunctions used for the pairing computation.

The following algorithm, Line_Function shows the computation of the line function. It takes Q_1 = (x_1, x_2), Q_2 = (x_2, y_2) in G_2, and P = (x, y) in G_1 as input, and outputs an element of G_T.

```
if (Q_1 = Q_2) then
    l := (3 * x_1^2) / (2 * y_1);
else if (Q_1 = -Q_2) then
    return x - x_1;
else
    l := (y_2 - y_1) / (x_2 - x_1);
end if;
return (l * (x - x_1) + y_1 - y);
```

When implementing the line function, implementers should consider the isomorphism of E and its twist curve E' so that one can reduce the computational cost of operations in G_2 [CLN09][KIK17]. We note that Line_function does not consider such an isomorphism.

The computation of the optimal Ate pairing uses the Frobenius endomorphism. The p-power Frobenius endomorphism pi for a point Q = (x, y) over E' is pi(p, Q) = (x^p, y^p).

A.1. Optimal Ate Pairings over Barreto-Naehrig Curves

Let c = 6 * t + 2 for a parameter t and c_0, c_1, ... , c_L in {-1,0,1} such that the sum of c_i * 2^i (i = 0, 1, ..., L) equals c.

The following algorithm shows the computation of the optimal Ate pairing on BN curves. It takes P in G_1, Q in G_2, an integer c, c_0, ...,c_L in {-1,0,1} such that the sum of c_i * 2^i (i = 0, 1, ..., L) equals c, and the order r of G_1 as input, and outputs e(P, Q).

```
f := 1; T := Q;
if (c_L = -1) then
    T := -T;
end if
for i = L-1 downto 0
    f := f^2 * Line_function(T, T, P); T := T + T;
    if (c_i = 1) then
        f := f * Line_function(T, Q, P); T := T + Q;
    else if (c_i = -1) then
        f := f * Line_function(T, -Q, P); T := T - Q;
    end if
end for
Q_1 := pi(p, Q); Q_2 := pi(p, Q_1);
f := f * Line_function(T, Q_1, P); T := T + Q_1;
f := f * Line_function(T, -Q_2, P);
f := f^{(p^k - 1) / r}
return f;
```

A.2. Optimal Ate Pairings over Barreto-Lynn-Scott Curves

Let c = t for a parameter t and c_0, c_1, ... , c_L in {-1,0,1} such that the sum of c_i * 2^i (i = 0, 1, ..., L) equals c.

The following algorithm shows the computation of the optimal Ate pairing on Barreto-Lynn-Scott curves. It takes P in G_1, Q in G_2, an integer c, c_0, ...,c_L in {-1,0,1} such that the sum of c_i * 2^i (i = 0, 1, ..., L) equals c, and the order r of G_1 as input, and outputs e(P, Q).

```
f := 1; T := Q;
if (c_L = -1) then
    T := -T;
end if
for i = L-1 downto 0
    f := f^2 * Line_function(T, T, P); T := T + T;
    if (c_i = 1) then
        f := f * Line_function(T, Q, P); T := T + Q;
    else if (c_i = -1) then
        f := f * Line_function(T, -Q, P); T := T - Q;
    end if
end for
f := f^{(p^k - 1) / r};
return f;
```

Appendix B. Test Vectors of Optimal Ate Pairing

We provide test vectors for Optimal Ate Pairing e(P, Q) given in Appendix A for the curves BLS12_381, BN462 and BLS48_581 given in Section 4. Here, the inputs P = (x, y) and Q = (x', y') are the corresponding base points BP and BP' given in Section 4.

For BLS12_381 and BN462, Q = (x', y') is given by

```
x' = x'_0 + x'_1 * u and
y' = y'_0 + y'_1 * u,
```

where u is an indeterminate and x'_0, x'_1, y'_0, y'_1 are elements of GF(p).

For BLS48_581, Q = (x', y') is given by

```
x' = x'_0 + x'_1 * u + x'_2 * v + x'_3 * u * v
    + x'_4 * w + x'_5 * u * w + x'_6 * v * w + x'_7 * u * v * w and
y' = y'_0 + y'_1 * u + y'_2 * v + y'_3 * u * v
    + y'_4 * w + y'_5 * u * w + y'_6 * v * w + y'_7 * u * v * w,
```

where u, v and w are indeterminates and x'_0, ..., x'_7 and y'_0, ..., y'_7 are elements of GF(p). The representation of Q = (x', y') given below is followed by [I-D.ietf-lwig-curve-representations].

In addition, we use the notation e_i (i = 0, ..., k-1) to represent each element in e(P, Q), where the extension field that e(P, Q) belongs is constructed according to [I-D.ietf-lwig-curve-representations].

BLS12_381:

Input x value:
0x17f1d3a73197d7942695638c4fa9ac0fc3688c4f9774b905a14e3a3f171bac58
6c55e83ff97a1aeffb3af00adb22c6bb

Input y value:
0x08b3f481e3aaa0f1a09e30ed741d8ae4fcf5e095d5d00af600db18cb2c04b3ed
d03cc744a2888ae40caa232946c5e7e1

Input x'_0 value:
0x024aa2b2f08f0a91260805272dc51051c6e47ad4fa403b02b4510b647ae3d177
0bac0326a805bbefd48056c8c121bdb8

Input x'_1 value:
0x13e02b6052719f607dacd3a088274f65596bd0d09920b61ab5da61bbdc7f5049
334cf11213945d57e5ac7d055d042b7e

Input y'_0 value:
0x0ce5d527727d6e118cc9cdc6da2e351aadfd9baa8cbdd3a76d429a695160d12c
923ac9cc3baca289e193548608b82801

Input y'_1 value:
0x0606c4a02ea734cc32acd2b02bc28b99cb3e287e85a763af267492ab572e99ab
3f370d275cec1da1aaa9075ff05f79be

e_0:
0x11619b45f61edfe3b47a15fac19442526ff489dcda25e59121d9931438907dfd
448299a87dde3a649bdba96e84d54558

e_1:
0x153ce14a76a53e205ba8f275ef1137c56a566f638b52d34ba3bf3bf22f277d70
f76316218c0dfd583a394b8448d2be7f

e_2:
0x095668fb4a02fe930ed44767834c915b283b1c6ca98c047bd4c272e9ac3f3ba6
ff0b05a93e59c71fba77bce995f04692

e_3:
0x16deedaa683124fe7260085184d88f7d036b86f53bb5b7f1fc5e248814782065
413e7d958d17960109ea006b2afdeb5f

e_4:
0x09c92cf02f3cd3d2f9d34bc44eee0dd50314ed44ca5d30ce6a9ec0539be7a86b
121edc61839ccc908c4bdde256cd6048

e_5:
0x111061f398efc2a97ff825b04d21089e24fd8b93a47e41e60eae7e9b2a38d54f
a4dedced0811c34ce528781ab9e929c7

e_6:
0x01ecfcf31c86257ab00b4709c33f1c9c4e007659dd5ffc4a735192167ce19705
8cfb4c94225e7f1b6c26ad9ba68f63bc

e_7:
0x08890726743a1f94a8193a166800b7787744a8ad8e2f9365db76863e894b7a11
d83f90d873567e9d645ccf725b32d26f

e_8:
0x0e61c752414ca5dfd258e9606bac08daec29b3e2c57062669556954fb227d3f1
260eedf25446a086b0844bcd43646c10

e_9:
0x0fe63f185f56dd29150fc498bbeea78969e7e783043620db33f75a05a0a2ce5c
442beaff9da195ff15164c00ab66bdde

```
   e_10:
      0x10900338a92ed0b47af211636f7cfdec717b7ee43900eee9b5fc24f0000c5874
      d4801372db478987691c566a8c474978

   e_11:
      0x1454814f3085f0e6602247671bc408bbce2007201536818c901dbd4d2095dd86
      c1ec8b888e59611f60a301af7776be3d

   BN462:

   Input x value:
      0x21a6d67ef250191fadba34a0a30160b9ac9264b6f95f63b3edbec3cf4b2e689d
      b1bbb4e69a416a0b1e79239c0372e5cd70113c98d91f36b6980d

   Input y value:
      0x0118ea0460f7f7abb82b33676a7432a490eeda842cccfa7d788c659650426e6a
      f77df11b8ae40eb80f475432c66600622ecaa8a5734d36fb03de

   Input x'_0 value:
      0x0257ccc85b58dda0dfb38e3a8cbdc5482e0337e7c1cd96ed61c913820408208f
      9ad2699bad92e0032ae1f0aa6a8b48807695468e3d934ae1e4df

   Input x'_1 value:
      0x1d2e4343e8599102af8edca849566ba3c98e2a354730cbed9176884058b18134
      dd86bae555b783718f50af8b59bf7e850e9b73108ba6aa8cd283

   Input y'_0 value:
      0x0a0650439da22c1979517427a20809eca035634706e23c3fa7a6bb42fe810f13
      99a1f41c9ddae32e03695a140e7b11d7c3376e5b68df0db7154e

   Input y'_1 value:
      0x073ef0cbd438cbe0172c8ae37306324d44d5e6b0c69ac57b393f1ab370fd725c
      c647692444a04ef87387aa68d53743493b9eba14cc552ca2a93a

   e_0:
      0x0cf7f0f2e01610804272f4a7a24014ac085543d787c8f8bf07059f93f87ba7e2
      a4ac77835d4ff10e78669be39cd23cc3a659c093dbe3b9647e8c

   e_1:
      0x00ef2c737515694ee5b85051e39970f24e27ca278847c7cfa709b0df408b830b
      3763b1b001f1194445b62d6c093fb6f77e43e369edefb1200389

   e_2:
      0x04d685b29fd2b8faedacd36873f24a06158742bb2328740f93827934592d6f17
      23e0772bb9ccd3025f88dc457fc4f77dfef76104ff43cd430bf7
```

e_3:
0x090067ef2892de0c48ee49cbe4ff1f835286c700c8d191574cb424019de11142
b3c722cc5083a71912411c4a1f61c00d1e8f14f545348eb7462c

e_4:
0x1437603b60dce235a090c43f5147d9c03bd63081c8bb1ffa7d8a2c31d6732308
60bb3dfe4ca85581f7459204ef755f63cba1fbd6a4436f10ba0e

e_5:
0x13191b1110d13650bf8e76b356fe776eb9d7a03fe33f82e3fe5732071f305d20
1843238cc96fd0e892bc61701e1844faa8e33446f87c6e29e75f

e_6:
0x07b1ce375c0191c786bb184cc9c08a6ae5a569dd7586f75d6d2de2b2f075787e
e5082d44ca4b8009b3285ecae5fa521e23be76e6a08f17fa5cc8

e_7:
0x05b64add5e49574b124a02d85f508c8d2d37993ae4c370a9cda89a100cdb5e1d
441b57768dbc68429ffae243c0c57fe5ab0a3ee4c6f2d9d34714

e_8:
0x0fd9a3271854a2b4542b42c55916e1faf7a8b87a7d10907179ac7073f6a1de04
4906ffaf4760d11c8f92df3e50251e39ce92c700a12e77d0adf3

e_9:
0x17fa0c7fa60c9a6d4d8bb9897991efd087899edc776f33743db921a689720c82
257ee3c788e8160c112f18e841a3dd9a79a6f8782f771d542ee5

e_10:
0x0c901397a62bb185a8f9cf336e28cfb0f354e2313f99c538cdceedf8b8aa22c2
3b896201170fc915690f79f6ba75581f1b76055cd89b7182041c

e_11:
0x20f27fde93cee94ca4bf9ded1b1378c1b0d80439eeb1d0c8daef30db0037104a
5e32a2ccc94fa1860a95e39a93ba51187b45f4c2c50c16482322

BLS48_581:

Input x value:
0x02af59b7ac340f2baf2b73df1e93f860de3f257e0e86868cf61abdbaedffb9f7
544550546a9df6f9645847665d859236ebdbc57db368b11786cb74da5d3a1e6d8c
3bce8732315af640

Input y value:
0x0cefda44f6531f91f86b3a2d1fb398a488a553c9efeb8a52e991279dd41b720e
f7bb7beffb98aee53e80f678584c3ef22f487f77c2876d1b2e35f37aef7b926b57
6dbb5de3e2587a70

x'_0:
0x05d615d9a7871e4a38237fa45a2775debabbefc70344dbccb7de64db3a2ef156
c46ff79baad1a8c42281a63ca0612f400503004d80491f510317b79766322154de
c34fd0b4ace8bfab

x'_1:
0x07c4973ece2258512069b0e86abc07e8b22bb6d980e1623e9526f6da12307f4e
1c3943a00abfedf16214a76affa62504f0c3c7630d979630ffd75556a01afa143f
1669b36676b47c57

x'_2:
0x01fccc70198f1334e1b2ea1853ad83bc73a8a6ca9ae237ca7a6d6957ccbab5ab
6860161c1dbd19242ffae766f0d2a6d55f028cbdfbb879d5fea8ef4cded6b3f0b4
6488156ca55a3e6a

x'_3:
0x0be2218c25ceb6185c78d8012954d4bfe8f5985ac62f3e5821b7b92a393f8be0
cc218a95f63e1c776e6ec143b1b279b9468c31c5257c200ca52310b8cb4e80bc3f
09a7033cbb7feafe

x'_4:
0x038b91c600b35913a3c598e4caa9dd63007c675d0b1642b5675ff0e7c5805386
699981f9e48199d5ac10b2ef492ae589274fad55fc1889aa80c65b5f746c9d4cbb
739c3a1c53f8cce5

x'_5:
0x0c96c7797eb0738603f1311e4ecda088f7b8f35dcef0977a3d1a58677bb03741
8181df63835d28997eb57b40b9c0b15dd7595a9f177612f097fc7960910fce3370
f2004d914a3c093a

x'_6:
0x0b9b7951c6061ee3f0197a498908aee660dea41b39d13852b6db908ba2c0b7a4
49cef11f293b13ced0fd0caa5efcf3432aad1cbe4324c22d63334b5b0e205c3354
e41607e60750e057

x'_7:
0x0827d5c22fb2bdec5282624c4f4aaa2b1e5d7a9defaf47b5211cf741719728a7
f9f8cfca93f29cff364a7190b7e2b0d4585479bd6aebf9fc44e56af2fc9e97c3f8
4e19da00fbc6ae34

y'_0:
0x00eb53356c375b5dfa497216452f3024b918b4238059a577e6f3b39ebfc435fa
ab0906235afa27748d90f7336d8ae5163c1599abf77eea6d659045012ab12c0ff3
23edd3fe4d2d7971

y’_1:
0x0284dc75979e0ff144da6531815fcadc2b75a422ba325e6fba01d72964732fcb
f3afb096b243b1f192c5c3d1892ab24e1dd212fa097d760e2e588b423525ffc7b1
11471db936cd5665

y’_2:
0x0b36a201dd008523e421efb70367669ef2c2fc5030216d5b119d3a480d370514
475f7d5c99d0e90411515536ca3295e5e2f0c1d35d51a652269cbc7c46fc3b8fde
68332a526a2a8474

y’_3:
0x0aec25a4621edc0688223fbbd478762b1c2cded3360dcee23dd8b0e710e122d2
742c89b224333fa40dced2817742770ba10d67bda503ee5e578fb3d8b8a1e53373
16213da92841589d

y’_4:
0x0d209d5a223a9c46916503fa5a88325a2554dc541b43dd93b5a959805f112985
7ed85c77fa238cdce8a1e2ca4e512b64f59f430135945d137b08857fdddfcf7a43
f47831f982e50137

y’_5:
0x07d0d03745736b7a513d339d5ad537b90421ad66eb16722b589d82e2055ab750
4fa83420e8c270841f6824f47c180d139e3aafc198caa72b679da59ed8226cf3a5
94eedc58cf90bee4

y’_6:
0x0896767811be65ea25c2d05dfdd17af8a006f364fc0841b064155f14e4c819a6
df98f425ae3a2864f22c1fab8c74b2618b5bb40fa639f53dccc9e884017d9aa62b
3d41faeafeb23986

y’_7:
0x035e2524ff89029d393a5c07e84f981b5e068f1406be8e50c87549b6ef8eca9a
9533a3f8e69c31e97e1ad0333ec719205417300d8c4ab33f748e5ac66e84069c55
d667ffcb732718b6

e_0:
0x0e26c3fcb8ef67417814098de5111ffcccc1d003d15b367bad07cef2291a93d3
1db03e3f03376f3beae2bd877bcfc22a25dc51016eda1ab56ee3033bc4b4fec596
2f02dffb3af5e38e

e_1:
0x069061b8047279aa5c2d25cdf676ddf34eddbc8ec2ec0f03614886fa828e1fc0
66b26d35744c0c38271843aa4fb617b57fa9eb4bd256d17367914159fc18b10a10
85cb626e5bedb145

e_2:
0x02b9bece645fbf9d8f97025a1545359f6fe3ffab3cd57094f862f7fb9ca01c88
705c26675bcc723878e943da6b56ce25d063381fcd2a292e0e7501fe572744184f
b4ab4ca071a04281

e_3:
0x0080d267bf036c1e61d7fc73905e8c630b97aa05ef3266c82e7a111072c0d205
6baa8137fba111c9650dfb18cb1f43363041e202e3192fced29d2b0501c882543f
b370a56bfdc2435b

e_4:
0x03c6b4c12f338f9401e6a493a405b33e64389338db8c5e592a8dd79eac7720dd
83dd6b0c189eeda20809160cd57cdf3e2edc82db15f553c1f6c953ea27114cb6bd
8a38e273f407dae0

e_5:
0x016e46224f28bfd8833f76ac29ee6e406a9da1bde55f5e82b3bd977897a9104f
18b9ee41ea9af7d4183d895102950a12ce9975669db07924e1b432d9680f5ce7e5
c67ed68f381eba45

e_6:
0x008ddce7a4a1b94be5df3ceea56bef0077dcdde86d579938a50933a47296d337
b7629934128e2457e24142b0eeaa978fd8e70986d7dd51fccbbeb8a1933434fec4
f5bc538de2646e90

e_7:
0x060ef6eae55728e40bd4628265218b24b38cdd434968c14bfefb87f0dcbfc76c
c473ae2dc0cac6e69dfdf90951175178dc75b9cc08320fcde187aa58ea047a2ee0
0b1968650eec2791

e_8:
0x0c3943636876fd4f9393414099a746f84b2633dfb7c36ba6512a0b48e66dcb2e
409f1b9e150e36b0b4311165810a3c721525f0d43a021f090e6a27577b42c7a57b
ed3327edb98ba8f8

e_9:
0x02d31eb8be0d923cac2a8eb6a07556c8951d849ec53c2848ee78c5eed40262eb
21822527a8555b071f1cd080e049e5e7ebfe2541d5b42c1e414341694d6f16d287
e4a8d28359c2d2f9

e_10:
0x07f19673c5580d6a10d09a032397c5d425c3a99ff1dd0abe5bec40a0d47a6b8d
aabb22edb6b06dd8691950b8f23faefcdd80c45aa3817a840018965941f4247f9f
97233a84f58b262e

e_11:
0x0d3fe01f0c114915c3bdf8089377780076c1685302279fd9ab12d07477aac03b
69291652e9f179baa0a99c38aa8851c1d25ffdb4ded2c8fe8b30338c14428607d6
d822610d41f51372

e_12:
0x0662eefd5fab9509aed968866b68cff3bc5d48ecc8ac6867c212a2d82cee5a68
9a3c9c67f1d611adac7268dc8b06471c0598f7016ca3d1c01649dda4b43531cffc
4eb41e691e27f2eb

e_13:
0x0aad8f4a8cfdca8de0985070304fe4f4d32f99b01d4ea50d9f7cd2abdc0aeea9
9311a36ec6ed18208642cef9e09b96795b27c42a5a744a7b01a617a91d9fb7623d
636640d61a6596ec

e_14:
0x0ffcf21d641fd9c6a641a749d80cab1bcad4b34ee97567d905ed9d5cfb74e9ae
f19674e2eb6ce3dfb706aa814d4a228db4fcd707e571259435393a27cac68b59a1
b690ae8cde7a94c3

e_15:
0x0cbe92a53151790cece4a86f91e9b31644a86fc4c954e5fa04e707beb69fc60a
858fed8ebd53e4cfd51546d5c0732331071c358d721ee601bfd3847e0e904101c6
2822dd2e4c7f8e5c

e_16:
0x0202db83b1ff33016679b6cfc8931deea6df1485c894dcd113bacf564411519a
42026b5fda4e16262674dcb3f089cd7d552f8089a1fec93e3db6bca43788cdb06f
c41baaa5c5098667

e_17:
0x070a617ed131b857f5b74b625c4ef70cc567f619defb5f2ab67534a1a8aa7297
5fc4248ac8551ce02b68801703971a2cf1cb934c9c354cadd5cfc4575cde8dbde6
122bd54826a9b3e9

e_18:
0x070e1ebce457c141417f88423127b7a7321424f64119d5089d883cb953283ee4
e1f2e01ffa7b903fe7a94af4bb1acb02ca6a36678e41506879069cee11c9dcf6a0
80b6a4a7c7f21dc9

e_19:
0x058a06be5a36c6148d8a1287ee7f0e725453fa1bb05cf77239f235b417127e37
0cfa4f88e61a23ea16df3c45d29c203d04d09782b39e9b4037c0c4ac8e8653e7c5
33ad752a640b233e

```
e_20:
   0x0dfdfaaeb9349cf18d21b92ad68f8a7ecc509c35fcd4b8abeb93be7a204ac871
   f2195180206a2c340fccb69dbc30b9410ed0b122308a8fc75141f673ae5ec82b6a
   45fc2d664409c6b6

e_21:
   0x0d06c8adfdd81275da2a0ce375b8df9199f3d359e8cf50064a3dc10a59241712
   4a3b705b05a7ffe78e20f935a08868ecf3fc5aba0ace7ce4497bb59085ca277c16
   b3d53dd7dae5c857

e_22:
   0x0708effd28c4ae21b6969cb9bdd0c27f8a3e341798b6f6d4baf27be259b4a476
   88b50cb68a69a917a4a1faf56cec93f69ac416512c32e9d5e69bd8836b6c2ba9c6
   889d507ad571dbc4

e_23:
   0x09da7c7aa48ce571f8ece74b98431b14ae6fb4a53ae979cd6b2e82320e8d25a0
   ece1ca1563aa5aa6926e7d608358af8399534f6b00788e95e37ef1b549f43a58ad
   250a71f0b2fdb2bf

e_24:
   0x0a7150a14471994833d89f41daeaa999dfc24a9968d4e33d88ed9e9f07aa2432
   c53e486ba6e3b6e4f4b8d9c989010a375935c06e4b8d6c31239fad6a61e2647b84
   a0e3f76e57005ff7

e_25:
   0x084696f31ff27889d4dccdc4967964a5387a5ae071ad391c5723c9034f16c255
   7915ada07ec68f18672b5b2107f785c15ddf9697046dc633b5a23cc0e442d28ef6
   eea9915d0638d4d8

e_26:
   0x0398e76e3d2202f999ac0f73e0099fe4e0fe2de9d223e78fc65c56e209cdf48f
   0d1ad8f6093e924ce5f0c93437c11212b7841de26f9067065b1898f48006bcc6f2
   ab8fa8e0b93f4ba4

e_27:
   0x06d683f556022368e7a633dc6fe319fd1d4fc0e07acff7c4d4177e83a911e733
   13e0ed980cd9197bd17ac45942a65d90e6cb9209ede7f36c10e009c9d337ee97c4
   068db40e34d0e361

e_28:
   0x0d764075344b70818f91b13ee445fd8c1587d1c0664002180bbac9a396ad4a8d
   c1e695b0c4267df4a09081c1e5c256c53fd49a73ffc817e65217a44fc0b20ef5ee
   92b28d4bc3e38576
```

e_29:
0x0aa6a32fdc4423b1c6d43e5104159bcd8e03a676d055d4496f7b1bc8761164a2
908a3ff0e4c4d1f4362015c14824927011e2909531b8d87ee0acd676e7221a1ca1
c21a33e2cf87dc51

e_30:
0x1147719959ac8eeab3fc913539784f1f947df47066b6c0c1beafecdb5fa784c3
be9de5ab282a678a2a0cbef8714141a6c8aaa76500819a896b46af20509953495e
2a85eff58348b38d

e_31:
0x11a377bcebd3c12702bb34044f06f8870ca712fb5caa6d30c48ace96898fcbcd
dbcf31f331c9e524684c02c90db7f30b9fc470d6e651a7e8b1f684383f3705d7a4
7a1b4fe463d623c8

e_32:
0x0b8b4511f451ba2cc58dc28e56d5e1d0a8f557ecb242f4d994a627e07cf3fa44
e6d83cb907deacf303d2f761810b5d943b46c4383e1435ec23fec196a70e339461
73c78be3c75dfc83

e_33:
0x090962d632ee2a57ce4208052ce47a9f76ea0fdad724b7256bb07f3944e9639a
981d3431087241e30ae9bf5e2ea32af323ce7ed195d383b749cb25bc09f678d385
a49a0c09f6d9efca

e_34:
0x0931c7befc80acd185491c68af886fa8ee39c21ed3ebd743b9168ae3b298df48
5bfdc75b94f0b21aecd8dca941dfc6d1566cc70dc648e6ccc73e4cbf2a1ac83c82
94d447c66e74784d

e_35:
0x020ac007bf6c76ec827d53647058aca48896916269c6a2016b8c06f0130901c8
975779f1672e581e2dfdbcf504e96ecf6801d0d39aad35cf79fbe7fe193c6c882c
15bce593223f0c7c

e_36:
0x0c0aed0d890c3b0b673bf4981398dcbf0d15d36af6347a39599f3a2258418482
8f78f91bbbbd08124a97672963ec313ff142c456ec1a2fc3909fd4429fd699d827
d48777d3b0e0e699

e_37:
0x0ef7799241a1ba6baaa8740d5667a1ace50fb8e63accc3bc30dc07b11d78dc54
5b68910c027489a0d842d1ba3ac406197881361a18b9fe337ff22d730fa44afabb
9f801f759086c8e4

e_38:
0x016663c940d062f4057257c8f4fb9b35e82541717a34582dd7d55b41ebadf40d
486ed74570043b2a3c4de29859fdeae9b6b456cb33bb401ecf38f9685646692300
517e9b035d6665fc

e_39:
0x1184a79510edf25e3bd2dc793a5082fa0fed0d559fa14a5ce9ffca4c61f17196
e1ffbb84326272e0d079368e9a735be1d05ec80c20dc6198b50a22a765defdc151
d437335f1309aced

e_40:
0x120e47a747d942a593d202707c936dafa6fed489967dd94e48f317fd3c881b10
41e3b6bbf9e8031d44e39c1ab5ae41e487eac9acd90e869129c38a8e6c97cf55d6
666d22299951f91a

e_41:
0x026b6e374108ecb2fe8d557087f40ab7bac8c5af0644a655271765d57ad71742
aa331326d871610a8c4c30ccf5d8adbeec23cdff20d9502a5005fce2593caf0682
c82e4873b89d6d71

e_42:
0x041be63a2fa643e5a66faeb099a3440105c18dca58d51f74b3bf281da4e689b1
3f365273a2ed397e7b1c26bdd4daade710c30350318b0ae9a9b16882c29fe31ca3
b884c92916d6d07a

e_43:
0x124018a12f0f0af881e6765e9e81071acc56ebcddadcd107750bd8697440cc16
f190a3595633bb8900e6829823866c5769f03a306f979a3e039e620d6d2f576793
d36d840b168eeedd

e_44:
0x0d422de4a83449c535b4b9ece586754c941548f15d50ada6740865be9c0b0667
88b6078727c7dee299acc15cbdcc7d51cdc5b17757c07d9a9146b01d2fdc7b8c56
2002da0f9084bde5

e_45:
0x1119f6c5468bce2ec2b450858dc073fea4fb05b6e83dd20c55c9cf694cbcc57f
c0effb1d33b9b5587852d0961c40ff114b7493361e4cfdff16e85fbce667869b6f
7e9eb804bcec46db

e_46:
0x061eaa8e9b0085364a61ea4f69c3516b6bf9f79f8c79d053e646ea637215cf65
90203b275290872e3d7b258102dd0c0a4a310af3958165f2078ff9dc3ac9e995ce
5413268d80974784

e_47:
0x0add8d58e9ec0c9393eb8c4bc0b08174a6b421e15040ef558da58d241e5f906a
d6ca2aa5de361421708a6b8ff6736efbac6b4688bf752259b4650595aa395c40d0
0f4417f180779985

Appendix C. ZCash serialization format for BLS12_381

This section describes the serialization format defined by [ZCashRep]. It is not officially standardized by the standards organization, however we show it in this appendix as a useful reference for implementers. This format applies to points on the BLS12_381 elliptic curves E and E', whose parameters are given in Section 4.2.1. Note that this serialization method is based on the representation shown in [SEC1] and it is a tiny tweak so as to apply to GF(p^m).

At a high level, the serialization format is defined as follows:

* Serialized points include three metadata bits that indicate whether a point is compressed or not, whether a point is the point at infinity or not, and (for compressed points) the sign of the point's y-coordinate.

* Points on E are serialized into 48 bytes (compressed) or 96 bytes (uncompressed). Points on E' are serialized into 96 bytes (compressed) or 192 bytes (uncompressed).

* The serialization of a point at infinity comprises a string of zero bytes, except that the metadata bits may be nonzero.

* The serialization of a compressed point other than the point at infinity comprises a serialized x-coordinate.

* The serialization of an uncompressed point other than the point at infinity comprises a serialized x-coordinate followed by a serialized y-coordinate.

Below, we give detailed serialization and de-serialization procedures. The following notation is used in the rest of this section:

* Elements of GF(p^2) are represented as polynomial with GF(p) coefficients like Section 2.5.

* For a byte string str, str[0] is defined as the first byte of str.

* The function sign_GF_p(y) returns one bit representing the sign of an element of GF(p). This function is defined as follows:

```
sign_GF_p(y) := { 1 if y > (p - 1) / 2, else
                { 0 otherwise.
```

* The function sign_GF_p^2(y') returns one bit representing the sign of an element in GF(p^2). This function is defined as follows:

```
sign_GF_p^2(y') := { sign_GF_p(y'_0) if y'_1 equals 0, else
                   { 1 if y'_1 > (p - 1) / 2, else
                   { 0 otherwise.
```

C.1. Point Serialization Procedure

The serialization procedure is defined as follows for a point P = (x, y). This procedure uses the I2OSP function defined in [RFC8017].

1. Compute the metadata bits C_bit, I_bit, and S_bit, as follows:

   * C_bit is 1 if point compression should be used, otherwise it is 0.

   * I_bit is 1 if P is the point at infinity, otherwise it is 0.

   * S_bit is 0 if P is the point at infinity or if point compression is not used. Otherwise (i.e., when point compression is used and P is not the point at infinity), if P is a point on E, S_bit = sign_GF_p(y), else if P is a point on E', S_bit = sign_GF_p^2(y).

2. Let m_byte = (C_bit * 2^7) + (I_bit * 2^6) + (S_bit * 2^5).

3. Let x_string be the serialization of x, which is defined as follows:

   * If P is the point at infinity on E, let x_string = I2OSP(0, 48).

   * If P is a point on E other than the point at infinity, then x is an element of GF(p), i.e., an integer in the inclusive range [0, p - 1]. In this case, let x_string = I2OSP(x, 48).

   * If P is the point at infinity on E', let x_string = I2OSP(0, 96).

   * If P is a point on E' other than the point at infinity, then x can be represented as (x_0, x_1) where x_0 and x_1 are elements of GF(p), i.e., integers in the inclusive range [0, p - 1] (see discussion of vector representations above). In this case, let x_string = I2OSP(x_1, 48) || I2OSP(x_0, 48).

Notice that in all of the above cases, the 3 most significant bits of x_string[0] are guaranteed to be 0.

4. If point compression is used, let y_string be the empty string. Otherwise (i.e., when point compression is not used), let y_string be the serialization of y, which is defined in Step 3.

5. Let s_string = x_string || y_string.

6. Set s_string[0] = x_string[0] OR m_byte, where OR is computed bitwise. After this operation, the most significant bit of s_string[0] equals C_bit, the next bit equals I_bit, and the next equals S_bit. (This is true because the three most significant bits of x_string[0] are guaranteed to be zero, as discussed above.)

7. Output s_string.

C.2. Point deserialization procedure

The deserialization procedure is defined as follows for a string s_string. This procedure uses the OS2IP function defined in [RFC8017].

1. Let m_byte = s_string[0] AND 0xE0, where AND is computed bitwise. In other words, the three most significant bits of m_byte equal the three most significant bits of s_string[0], and the remaining bits are 0.

   If m_byte equals any of 0x20, 0x60, or 0xE0, output INVALID and stop decoding.

   Otherwise:

   * Let C_bit equal the most significant bit of m_byte,

   * Let I_bit equal the second most significant bit of m_byte, and

   * Let S_bit equal the third most significant bit of m_byte.

2. If C_bit is 1:

   * If s_string has length 48 bytes, the output point is on the curve E.

   * If s_string has length 96 bytes, the output point is on the curve E’.

   * If s_string has any other length, output INVALID and stop decoding.

   If C_bit is 0:

   * If s_string has length 96 bytes, the output point is on E.

   * If s_string has length 192 bytes, the output point is on E'.

   * If s_string has any other length, output INVALID and stop decoding.

3. Let s_string[0] = s_string[0] AND 0x1F, where AND is computed bitwise. In other words, set the three most significant bits of s_string[0] to 0.

4. If I_bit is 1:

   * If s_string is not the all zeros string, output INVALID and stop decoding.

   * Otherwise (i.e., if s_string is the all zeros string), output the point at infinity on the curve that was determined in step 2 and stop decoding.

   Otherwise, I_bit must be 0. Continue decoding.

5. If C_bit is 0:

   * Let x_string be the first half of s_string.

   * Let y_string be the last half of s_string.

   * Let x = OS2IP(x_string).

   * Let y = OS2IP(y_string).

   * If the point P = (x, y) is not a valid point on the curve that was determined in step 2, output INVALID and stop decoding.

   * Otherwise, output the point P = (x, y) and stop decoding.

   Otherwise, C_bit must be 1. Continue decoding.

6. Let x = OS2IP(s_string).

7. If the curve that was determined in step 2 is E:

* Let y2 = x^3 + 4 in GF(p).

* If y2 is not square in GF(p), output INVALID and stop decoding.

* Otherwise, let y = sqrt(y2) in GF(p) and let Y_bit = sign_GF_p(y).

Otherwise, (i.e., when the curve that was determined in step 2 is E’):

* Let y2 = x^3 + 4 * (u + 1) in GF(p^2).

* If y2 is not square in GF(p^2), output INVALID and stop decoding.

* Otherwise, let y = sqrt(y2) in GF(p^2) and let Y_bit = sign_GF_p^2(y).

8. If S_bit equals Y_bit, output P = (x, y) and stop decoding. Otherwise, output P = (x, -y) and stop decoding.

Appendix D. Adoption Status of Pairing-Friendly Curves with the 100-bit Security Level

BN curves including BN254 that were estimated as the 128-bit security level before exTNFS ensure no more than the 100-bit security level by the effect of exTNFS. Table 2 summarizes the adoption status of the parameters with a security level lower than the "Arnd 128-bit" range. Please refer the Section 4 for the naming conventions for each curve listed in Table 2.

Category	Name	Supported 100-bit Curves
Standard	ISO/IEC	BN256I
	TCG	BN256I
	FIDO/W3C	BN256I
		BN256D
Library	mcl	BN254N
		BN_SNARK1
	TEPLA	BN254B

		BN254N
	RELIC	BN254N
		BN256D
	AMCL	BN254N
		BN254CX
		BN256I
	Intel IPP	BN256I
	MIRACL	BN254N
		BN254CX
		BN256I
	Adjoint	BN_SNARK1
		BN254B
		BN254N
		BN254S1
		BN254S2
Application	Zcash	BN_SNARK1
	DFINITY	BN254N
		BN_SNARK1

Table 2: Adoption Status of Pairing-Friendly Curves with 100-bit Security Level(Legacy)

Authors' Addresses

Yumi Sakemi (editor)
GMO CONNECT Inc.
Email: sakemi-yumi@gmo-connect.jp

Satoru Kanno
GMO CONNECT Inc.
Email: kanno@gmo-connect.jp

Riad S. Wahby
Stanford University
Email: rsw@cs.stanford.edu