Petrelic Documentation

Laurent Girod, Wouter Lueks

Feb 15, 2022

Basics

CHAPTER 1

petrelic

`petrelic` is a Python wrapper around RELIC. It provides a simple python interface to the BLS-381 pairing and RELIC's big number class. Our goals is to make it easy to prototype new cryptographic applications in Python using RELIC as the backend. In the future we aim to support a few other pairing curves as well.

`petrelic` provides native, multiplicative and additive interfaces to RELIC. You can use the one that you find most comfortable. `petrelic` overloads Python's binary operators to make computation with pairings easy. For example, here is how you would compute and verify a BLS signature using the multiplicative interface:

```
>>> from petrelic.multiplicative.pairing import G1, G2, GT
>>> sk = G1.order().random()
>>> pk = G2.generator() ** sk

>>> # Create the signature
>>> m = b"Some message"
>>> signature = G1.hash_to_point(m) ** sk

>>> # Verify the signature
>>> signature.pair(G2.generator()) == G1.hash_to_point(m).pair(pk)
True
```

You can find more information in the documentation.

You can install `petrelic` on Linux using:

```
$ pip install petrelic
```

For full details see the installation documentation.

Warning: Please don't use this software for anything mission-critical. It is designed for rapid prototyping of cryptographic primitives using RELIC. We offer no guarantees that the `petrelic` bindings are secure. We echo RELIC own warning: "RELIC is at most alpha-quality software. Implementations may not be correct or secure and may include patented algorithms. . . . Use at your own risk."

1.1 Getting started

1.1.1 petrelic

`petrelic` is a Python wrapper around RELIC. It provides a simple python interface to the BLS-381 pairing and RELIC's big number class. Our goals is to make it easy to prototype new cryptographic applications in Python using RELIC as the backend. In the future we aim to support a few other pairing curves as well.

`petrelic` provides native, multiplicative and additive interfaces to RELIC. You can use the one that you find most comfortable. `petrelic` overloads Python's binary operators to make computation with pairings easy. For example, here is how you would compute and verify a BLS signature using the multiplicative interface:

```
>>> from petrelic.multiplicative.pairing import G1, G2, GT
>>> sk = G1.order().random()
>>> pk = G2.generator() ** sk

>>> # Create the signature
>>> m = b"Some message"
>>> signature = G1.hash_to_point(m) ** sk

>>> # Verify the signature
>>> signature.pair(G2.generator()) == G1.hash_to_point(m).pair(pk)
True
```

You can find more information in the documentation.

You can install `petrelic` on Linux using:

```
$ pip install petrelic
```

For full details see the installation documentation.

Warning: Please don't use this software for anything mission-critical. It is designed for rapid prototyping of cryptographic primitives using RELIC. We offer no guarantees that the `petrelic` bindings are secure. We echo RELIC own warning: "RELIC is at most alpha-quality software. Implementations may not be correct or secure and may include patented algorithms. . . . Use at your own risk."

1.2 Installation

You can install `petrelic` with `pip`:

```
pip install petrelic
```

1.2.1 Supported platforms

Currently, `petrelic` was tested successfully with python 3.7 on Debian 10 on x86_64 architecture. It might also work with Python 3.6 and 3.8, but no tests were performed for these versions.

1.2.2 Building petrelic on Linux

`petrelic` ships `manylinux2010` wheels which include all binary dependencies. For users running a `manylinux2010` compatible distribution (that is almost all distribution with a few exceptions, the most well known being Alpine), `petrelic` can be installed trivially with `pip`:

```
pip install petrelic
```

1.2.3 Full manual install

If you want to build `petrelic` fully manually, you will need to install RELIC with some custom compilation options.

The first step is to clone RELIC's Git repository, and to tweak the compilation options of the preset `x64-pbc-bls12-381.sh` before executing it:

```
git clone https://github.com/relic-toolkit/relic.git
cd relic/presets
sed 's/DSHLIB=OFF/DSHLIB=ON/' preset/x64-pbc-bls12-381.sh > preset/00custom.sh
cd ..
bash preset/00custom.sh -DCMAKE_INSTALL_PREFIX='/usr/local' .
```

Then RELIC can be build and installed with `make`:

```
make
sudo make install
```

Once installed, `petrelic` can be installed with `pip`:

```
git clone https://github.com/spring-epfl/petrelic.git
cd petrelic
pip3 install -v -e '.[dev]'
```

Potentially you'll need to set your library path:

```
export LD_LIBRARY_PATH=/usr/local/lib:"$LD_LIBRARY_PATH"
```

1.2.4 Building the wheels manually

If you are using a non `manylinux2010` compatible distribution, or if you prefer to compile `petrelic` yourself, you can do it by using the provided building script `build.sh`.

Running this script requires Docker. If Docker is not yet installed on your system, please refer to its documentation to install it.

The other typical building tools like `make` or `gcc` are already installed in the `manylinux2010` Docker image, therefore no more dependencies are required to build `petrelic`.

Once Docker is installed on your system, you can build the wheel by running:

```
sh build.sh
```

Once the script has finished running, which takes about 10 minutes, some wheels will be copied on your working directory from the working directory in `/tmp`, which can be installed with `pip`:

```
pip install petrelic-0.1.0-cp37-cp37m-manylinux2010_x86_64.whl
```

1.2.5 Zksk Integration

This library can be integrated with zksk, to do so, the `bn-wrapper` branch of zksk needs to be installed, and a global variable needs to be changed:

```
cd ..
git clone https://github.com/spring-epfl/zksk.git
cd zksk
git checkout bn-wrapper
sed -i 's/BACKEND\s*=\s*"openssl"/BACKEND = "relic"/' zksk/bn.py
cd ../petrelic
. venv/bin/activate
pip install -e ../zksk/
```

1.3 Structure

`petrelic` provides three interfaces, `petrelic.native`, `petrelic.multiplicative`, and `petrelic.additive` to `RELIC` BLS-381 curve. In addition, it provides a binding to `RELIC`'s big number (`Bn`) interface to ease integration between the two. In general, Python's integers can be substituted for `RELIC`'s big numbers, and will be automatically converted. See the reference for more details on how to use these interfaces.

1.3.1 RELIC Bignums

class `petrelic.bn.`**`Bn`**(*num=0*)

abs()

binary()

A byte array representing the absolute value

A byte array representation of the absolute value in Big-Endian format (with 8 bit atomic elements). You are responsible for extracting the sign bit separately.

Example:

```
>>> from binascii import hexlify
```

```
>>> bin = Bn(66051).binary()
>>> hexlify(bin) == b'010203'
True
```

```
>>> bin = Bn(1337).binary()
>>> hexlify(bin) == b'0539'
True
```

bn

bool()

Turn Bn into boolean. False if zero, True otherwise.

Examples:

```
>>> bool(Bn(0))
False
>>> bool(Bn(1337))
True
```

copy()

Returns a copy of the Bn object.

divmod(*other*)

Returns the integer division and remainder of this number by another.

Example:

```
>>> Bn(13).divmod(Bn(9))
(Bn(1), Bn(4))
```

static from_binary(*sbin*)

Restore number given its Big-endian representation.

Creates a Big Number from a byte sequence representing the number in Big-endian 8 byte atoms. Only positive values can be represented as byte sequence, and the library user should store the sign bit separately.

Args: sbin (string): a byte sequence.

Example:

```
>>> from binascii import unhexlify
>>> byte_seq = unhexlify(b"010203")
>>> Bn.from_binary(byte_seq)
Bn(66051)
>>> (1 * 256**2) + (2 * 256) + 3
66051
```

static from_decimal(*sdec*)

Creates a Big Number from a decimal string.

Args: sdec (string): numeric string possibly starting with minus.

See Also: str() produces a decimal string from a big number.

Example:

```
>>> hundred = Bn.from_decimal("100")
>>> str(hundred)
'100'
```

static from_hex(*shex*)

Creates a Big Number from a hexadecimal string.

Args: shex (string): hex (0-F) string possibly starting with minus.

See Also: hex() produces a hexadecimal representation of a big number.

Example:

```
>>> Bn.from_hex("FF")
Bn(255)
```

static from_num(*num*)

static get_prime(*bits*, *safe=1*)

Builds a prime Big Number of length bits.

Args: bits (int) – the number of bits. safe (int) – 1 for a safe prime, otherwise 0.

static get_random(*bits*)

Generates a random number of the given number of bits

Args: bits (int) – The number of bits

Examples:

```
>>> n = Bn.get_random(10)
>>> n.num_bits() <= 10
True
```

hex()

The representation of the string in hexadecimal. Synonym for hex(n).

int()

A native python integer representation of the Big Number. Synonym for int(bn).

int_add(*other*)

Returns the sum of this number with another. Synonym for self + other.

Example:

```
>>> one100 = Bn(100)
>>> two100 = Bn(200)
>>> two100.int_add(one100) # Function syntax
Bn(300)
>>> two100 + one100        # Operator syntax
Bn(300)
```

int_div(*other*)

Returns the integer division of this number by another. Synonym of self / other.

Example:

```
>>> one100 = Bn(100)
>>> two100 = Bn(200)
>>> two100.int_div(one100) # Function syntax
Bn(2)
>>> two100 // one100        # Operator syntax
Bn(2)
```

int_mul(*other*)

Returns the product of this number with another. Synonym for self * other.

Example:

```
>>> one100 = Bn(100)
>>> two100 = Bn(200)
>>> one100.int_mul(two100) # Function syntax
Bn(20000)
>>> one100 * two100         # Operator syntax
Bn(20000)
```

int_neg()

Returns the negative of this number. Synonym with -self.

Example:

```
>>> one100 = Bn(100)
>>> one100.int_neg()
Bn(-100)
>>> -one100
Bn(-100)
```

int_sub(*other*)

Returns the difference between this number and another. Synonym for self - other.

Example:

```
>>> one100 = Bn(100)
>>> two100 = Bn(200)
>>> two100.int_sub(one100) # Function syntax
Bn(100)
>>> two100 - one100        # Operator syntax
Bn(100)
```

is_bit_set(*n*)

Returns True if the nth bit is set

Examples:

```
>>> a = Bn(17) # in binary 10001
>>> a.is_bit_set(0)
True
>>> a.is_bit_set(1)
False
>>> a.is_bit_set(4)
True
```

is_even()

Returns True if the number is even.

Examples:

```
>>> Bn(2).is_even()
True
>>> Bn(1337).is_even()
False
```

is_odd()

Returns True if the number is odd.

Examples:

```
>>> Bn(2).is_odd()
False
>>> Bn(1337).is_odd()
True
```

is_prime()
Returns True if the number is prime, with negligible prob. of error.

Examples:

```
>>> Bn(37).is_prime()
True
>>> Bn(10).is_prime()
False
```

mod(*other*)
Returns the remainder of this number modulo another. Synonym for self % other.

Example:

```
>>> one100 = Bn(100)
>>> two100 = Bn(200)
>>> two100.mod(one100) # Function syntax
Bn(0)
>>> two100 % one100         # Operator syntax
Bn(0)
```

mod_add(*other*, *m*)
Returns the sum of self and other modulo m.

Example:

```
>>> Bn(10).mod_add(2, 11)
Bn(1)
>>> Bn(10).mod_add(Bn(2), Bn(11))
Bn(1)
```

mod_inverse(*m*)
Compute the inverse mod m, such that self * res == 1 mod m.

Example:

```
>>> Bn(10).mod_inverse(m = Bn(11))
Bn(10)
>>> Bn(10).mod_mul(Bn(10), m = Bn(11)) == Bn(1)
True
```

mod_mul(*other*, *m*)
Return the product of self and other modulo m.

Example:

```
>>> Bn(10).mod_mul(Bn(2), Bn(11))
Bn(9)
```

mod_pow(*other*, *m*, *ctx=None*)
Performs the modular exponentiation of self ** other % m.

This function is _not_ constant time.

Example:

```
>>> one100 = Bn(100)
>>> one100.mod_pow(2, 3)    # Modular exponentiation
Bn(1)
```

mod_sub (*other*, *m*)
Returns the difference of self and other modulo m.

Example:

```
>>> Bn(10).mod_sub(Bn(2), Bn(11))
Bn(8)
```

num_bits ()
Returns the number of bits representing this Big Number

pow (*other*, *modulo=None*)
Returns the number raised to the power other optionally modulo a third number. Synonym with pow(self, other, modulo).

Example:

```
>>> one100 = Bn(100)
>>> one100.pow(2)       # Function syntax
Bn(10000)
>>> one100 ** 2         # Operator syntax
Bn(10000)
>>> one100.pow(2, 3)    # Modular exponentiation
Bn(1)
```

random ()
Returns a random number 0 < rand < self

TODO: currently it excludes 0, update to include 0

Example: #>>> r = Bn(100).random() #>>> 0 <= r && r < 100 True

repr ()

repr_in_base (*radix*)
Represent number as string in given base

Args: radix (int): The number of unique digits (2 <= radix <= 62)

Examples:

```
>>> Bn(42).repr_in_base(16)
'2A'
>>> Bn(-1024).repr_in_base(2)
'-10000000000'
```

test ()

```
>>> b = Bn()
>>> b.repr_in_base(2)
'0'
```

1.3.2 Native interface

The native wrapper follows RELIC's convention of writing operations in G1 and G2 additively, and those in GT multiplicatively. You can use this interface by importing

```
from petrelic.native.pairing import G1, G2, GT
```

petrelic.native.pairing

This module provides a Python wrapper around RELIC's pairings using a native interface: operations in `petrelic.native.pairings.G1` and `petrelic.native.pairings.G2` are written additively, whereas operations in `petrelic.native.pairings.GT` are written multiplicatively.

Let's see how we can use this interface to implement the Boney-Lynn-Shacham signature scheme for type III pairings. First we generate a private key:

```
>>> sk = G1.order().random()
```

which is a random integer modulo the group order. Note that for this setting, all three groups have the same order. Next, we generate the corresponding public key:

```
>>> pk = (sk * G1.generator(), sk * G2.generator())
```

(For security in the type III setting, the first component is a necessary part of the public key. It is not actually used in the scheme.) To sign a message *m* we first hash it to the curve G1 using *`G1.hash_to_point()`* and then multiply it with the signing key *sk* to obtain a signature:

```
>>> m = b"Some message"
>>> signature = sk * G1.hash_to_point(m)
```

Finally, we can use the pairing operator to verify the signature:

```
>>> signature.pair(G2.generator()) == G1.hash_to_point(m).pair(pk[1])
True
```

Indeed, the pairing operator is bilinear. For example:

```
>>> a, b = 13, 29
>>> A = a * G1.generator()
>>> B = b * G2.generator()
>>> A.pair(B) == G1.generator().pair(G2.generator()) ** (a * b)
True
```

class `petrelic.native.pairing.`**`BilinearGroupPair`**

A bilinear group pair used to wrap the three groups G1, G2, GT.

`groups``()`

Returns the three groups in the following order : G1, G2, GT.

class `petrelic.native.pairing.`**`G1`**

The G1 group.

classmethod generator`()`

Return generator of the group.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> generator + neutral == generator
True
```

classmethod hash_to_point(*hinput*)
Return group element obtained by hashing the input

Example:

```
>>> elem = G1.hash_to_point(b"foo")
>>> elem.is_valid()
True
```

classmethod infinity()
The point at infinity.

Alias for *G1.neutral_element()*

classmethod neutral_element()
Return the neutral element of the group G1.

In this case, the point at infinity.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> generator + neutral == generator
True
```

classmethod order()
Return the order of the group as a Bn large integer.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> order = G1.order()
>>> order * generator == neutral
True
```

classmethod sum(*elems*)
Efficient sum of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ x * G1.generator() for x in [10, 25, 13]]
>>> G1.sum(elems) ==  (10 + 25 + 13) * G1.generator()
True
```

classmethod wsum(*weights, elems*)
Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ x * G1.generator() for x in [10, 25, 13]]
>>> G1.wsum(weights, elems) ==  (1 * 10 + 2 * 25 + 3 * 13) * G1.
↪generator()
True
```

class petrelic.native.pairing.**G1Element**

Element of the G1 group.

add(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 40 * G1.generator()
>>> a + b == 50 * G1.generator()
True
>>> a.add(b) == 50 * G1.generator()
True
```

double()

Return double of the current element

Example:

```
>>> generator = G1.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

eq(*other*)

Check point equality.

classmethod from_binary(*sbin*)

Deserialize a binary representation of the element of G1.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

get_affine_coordinates()

Return the affine coordinates (x,y) of this EC Point.

Example:

```
>>> generator = G1.generator()
>>> x, y = generator.get_affine_coordinates()
>>> x
Bn(368541675371338701678108831518307775796162079578254640989457837868860759237837631883
>>> y
Bn(133950654494447647302047137994192122158493387593834962042654373641651142395633350647
```

group
alias of *G1*

iadd(*other*)
Inplace add another point.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 10 * G1.generator()
>>> a += 3 * G1.generator()
>>> _ = b.iadd(3 * G1.generator())
>>> a == b
True
>>> a == 13 * G1.generator()
True
```

idouble()
Inplace double the current element.

Example:

```
>>> generator = G1.generator()
>>> elem = G1.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

iinverse()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G1.generator()
>>> elem2 = a * G1.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)
Inplace point multiplication by a scalar

Examples:

```
>>> a = G1.generator()
>>> b = G1.generator()
>>> a *= 10
>>> _ = b.imul(10)
>>> a == b
True
>>> a == 10 * G1.generator()
True
```

inverse()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G1.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G1.order() - a) * G1.generator()
True
```

is_infinity()

Check if the object is the neutral element of G1.

Example:

```
>>> generator = G1.generator()
>>> order = G1.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_neutral_element()

Check if the object is the neutral element of G1.

Example:

```
>>> generator = G1.generator()
>>> order = G1.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_valid()

Check if the element is a valid element on the curve. This method excludes the unity element. For that use *is_infinity*.

Example:

```
>>> elem = G1.hash_to_point(b"foo")
>>> elem.is_valid()
True
>>> elem = G1.infinity()
>>> elem.is_valid()
False
```

isub(*other*)

Inplace substract another point.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 10 * G1.generator()
>>> a -= 3 * G1.generator()
>>> _ = b.isub(3 * G1.generator())
>>> a == b
True
>>> a == 7 * G1.generator()
True
```

mul(*other*)

Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = G1.generator()
>>> g + g == 2 * g
True
```

ne (*other*)
Check that the points are different.

neg ()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G1.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G1.order() - a) * G1.generator()
True
```

pair (*other*)
Pair element with another element in G2

Computes the bilinear pairing between self and another element in *petrelic.native.pairing.G2*.

Examples:

```
>>> g1, g2 = G1.generator(), G2.generator()
>>> a, b = 10, 50
>>> A, B = a * g1, b * g2
>>> A.pair(B) == g1.pair(g2) ** (a * b)
True
```

```
>>> A.pair(g2) == g1.pair(a * g2)
True
>>> A.pair(g2) == g1.pair(g2) ** a
True
```

sub (*other*)
Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * G1.generator()
>>> b = 13 * G1.generator()
>>> a - b == 37 * G1.generator()
True
>>> a.sub(b) == 37 * G1.generator()
True
```

to_binary (*compressed=True*)
Serialize the element of G1 into a binary representation.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

class petrelic.native.pairing.**G2**

G2 group.

classmethod generator()

Return generator of the group.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> generator + neutral == generator
True
```

classmethod hash_to_point(*hinput*)

Return group element obtained by hashing the input

Example:

```
>>> elem = G2.hash_to_point(b"foo")
>>> elem.is_valid()
True
```

classmethod infinity()

The point at infinity.

Alias for *G1.neutral_element()*

classmethod neutral_element()

Return the neutral element of the group G2.

In this case, the point at infinity.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> generator + neutral == generator
True
```

classmethod order()

Return the order of the EC group as a Bn large integer.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> order = G2.order()
>>> order * generator == neutral
True
```

classmethod sum(*elems*)

Efficient sum of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ x * G2.generator() for x in [10, 25, 13]]
>>> G2.sum(elems) ==  (10 + 25 + 13) * G2.generator()
True
```

classmethod wsum(*weights, elems*)

Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ x * G2.generator() for x in [10, 25, 13]]
>>> G2.wsum(weights, elems) ==  (1 * 10 + 2 * 25 + 3 * 13) * G2.
↪generator()
True
```

class petrelic.native.pairing.**G2Element**

Element of the G2 group.

add(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 40 * G2.generator()
>>> a + b == 50 * G2.generator()
True
>>> a.add(b) == 50 * G2.generator()
True
```

double()

Return double of the current element

Example:

```
>>> generator = G2.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

eq(*other*)

Check that the points on the EC are equal.

classmethod from_binary(*sbin*)

Deserialize a binary representation of the element of G2.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

iadd(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 40 * G2.generator()
>>> a + b == 50 * G2.generator()
True
>>> a.add(b) == 50 * G2.generator()
True
```

idouble()

Inplace double the current element.

Example:

```
>>> generator = G2.generator()
>>> elem = G2.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

iinverse()

Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G2.generator()
>>> elem2 = a * G2.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)

Inplace point multiplication by a scalar

Examples:

```
>>> a = G2.generator()
>>> b = G2.generator()
>>> a *= 10
>>> _ = b.imul(10)
>>> a == b
True
>>> a == 10 * G2.generator()
True
```

inverse()

Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G2.generator()
>>> -elem == elem.inverse()
True
```

(continues on next page)

(continued from previous page)

```
>>> elem.inverse() == (G2.order() - a) * G2.generator()
True
```

is_infinity()
Check if the object is the neutral element of G2.

Example:

```
>>> generator = G2.generator()
>>> order = G2.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_neutral_element()
Check if the object is the neutral element of G2.

Example:

```
>>> generator = G2.generator()
>>> order = G2.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is a valid element on the curve. This method excludes the unity element. For that use *is_infinity*.

Example:

```
>>> elem = G2.hash_to_point(b"foo")
>>> elem.is_valid()
True
>>> elem = G2.infinity()
>>> elem.is_valid()
False
```

isub(*other*)
Inplace substract another point.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 10 * G2.generator()
>>> a -= 3 * G2.generator()
>>> _ = b.isub(3 * G2.generator())
>>> a == b
True
>>> a == 7 * G2.generator()
True
```

mul(*other*)
Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = G2.generator()
>>> g + g == 2 * g
True
```

ne(*other*)
Check that the points on the EC are not equal.

neg()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G2.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G2.order() - a) * G2.generator()
True
```

sub(*other*)
Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * G2.generator()
>>> b = 13 * G2.generator()
>>> a - b == 37 * G2.generator()
True
>>> a.sub(b) == 37 * G2.generator()
True
```

to_binary(*compressed=True*)
Serialize the element of G2 into a binary representation.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

class petrelic.native.pairing.**GT**
GT group.

classmethod generator()
Return generator of the EC group.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> generator * neutral == generator
True
```

classmethod neutral_element()
Return the neutral element of the group GT. In this case, the unity point.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> generator * neutral == generator
True
```

classmethod order()
Return the order of the EC group as a Bn large integer.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> order = GT.order()
>>> generator ** order == neutral
True
```

classmethod prod(*elems*)
Efficient product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ GT.generator() ** x for x in [10, 25, 13]]
>>> GT.prod(elems) ==  GT.generator() ** (10 + 25 + 13)
True
```

classmethod unity()
The unity elements

Alias for `GT.neutral_element()`

classmethod wprod(*weights, elems*)
Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ GT.generator() ** x for x in [10, 25, 13]]
>>> GT.wprod(weights, elems) ==  GT.generator() ** (1 * 10 + 2 * 25 + 3 *␣
↪13)
True
```

class petrelic.native.pairing.**GTElement**
GT element.

div(*other*)
Divide two points

This method is aliased by *a / b* and *a // b*.

Examples:

```
>>> a = GT.generator() ** 50
>>> b = GT.generator() ** 13
>>> a / b == GT.generator() ** 37
True
```

(continues on next page)

(continued from previous page)

```
>>> a // b == GT.generator() ** 37
True
>>> a.div(b) == GT.generator() ** 37
True
```

eq (*other*)
Check that the points are equal.

classmethod from_binary (*sbin*)
Deserialize a binary representation of the element of GT.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

group
alias of *GT*

idiv (*other*)
Inplace division by another point

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 10
>>> a /= GT.generator() ** 3
>>> _ = b.idiv(GT.generator() ** 3)
>>> a == b
True
>>> a == GT.generator() ** 7
True
```

iinverse ()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = GT.generator() ** a
>>> elem2 = GT.generator() ** a
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul (*other*)
Inplace multiplication by another element

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 10
>>> a *= GT.generator() ** 3
>>> _ = b.imul(GT.generator() ** 3)
>>> a == b
```

(continues on next page)

(continued from previous page)

```
True
>>> a == GT.generator() ** 13
True
```

inverse()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = GT.generator() ** a
>>> elem.inverse() == GT.generator() ** (G1.order() - a)
True
```

ipow(*other*)
Inplace raise element to the power of a scalar

Examples:

```
>>> g = GT.generator()
>>> a = GT.generator()
>>> _ = a.ipow(3)
>>> g * g * g == a
True
```

is_neutral_element()
Check if the object is the neutral element of GT.

Example:

```
>>> generator = GT.generator()
>>> order = GT.order()
>>> elem = generator ** order
>>> elem.is_neutral_element()
True
```

is_unity()
Check if the object is the neutral element of GT.

Example:

```
>>> generator = GT.generator()
>>> order = GT.order()
>>> elem = generator ** order
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is in the group

Example:

```
>>> elem = GT.generator() ** 1337
>>> elem.is_valid()
True
```

isquare()
Inplace square of the current element.

Example:

```
>>> elem = GT.generator()
>>> _ = elem.isquare()
>>> elem == GT.generator() ** 2
True
```

mul (*other*)

Multiply two elements

This method is aliased by *a * b*.

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 40
>>> a * b == GT.generator() ** 50
True
>>> a.mul(b) == GT.generator() ** 50
True
```

ne (*other*)

Check that the points on the EC are not equal.

pow (*other*)

Raise element to the power of a scalar

This method is aliased by *el ** n*.

Examples:

```
>>> g = GT.generator()
>>> g * g == g ** 2
True
>>> g * g == g.pow(2)
True
```

square ()

Return the square of the current element

Example:

```
>>> generator = GT.generator()
>>> elem = generator.square()
>>> elem == generator ** 2
True
```

to_binary (*compressed=True*)

Serialize the element of GT into a binary representation.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

exception petrelic.native.pairing.**NoAffineCoordinateForECPoint**

Exception raised when an EC point can not be represented as affine coordinates.

args

msg = 'No affine coordinates exists for the given EC point.'

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

1.3.3 Additive interface

The additive wrapper presents an additive interface for all three groups G1, G2, and GT. This is useful, for example, when integrating with *zsks*. You can use this interface by importing:

```
from petrelic.additive.pairing import G1, G2, GT
```

petrelic.additive.pairing

This module provides a Python wrapper around RELIC's pairings using a additive interface: operations in `petrelic.additive.pairings.G1`, `petrelic.additive.pairings.G2`, and `petrelic.additive.pairings.GT` are all written additively.

Let's see how we can use this interface to implement the Boney-Lynn-Shacham signature scheme for type III pairings. First we generate a private key:

```
>>> sk = G1.order().random()
```

which is a random integer modulo the group order. Note that for this setting, all three groups have the same order. Next, we generate the corresponding public key:

```
>>> pk = (sk * G1.generator(), sk * G2.generator())
```

(For security in the type III setting, the first component is a necessary part of the public key. It is not actually used in the scheme.) To sign a message *m* we first hash it to the curve G1 using *G1.hash_to_point()* and then raise it to the power of the signing key *sk* to obtain a signature:

```
>>> m = b"Some message"
>>> signature = sk * G1.hash_to_point(m)
```

Finally, we can use the pairing operator to verify the signature:

```
>>> signature.pair(G2.generator()) == G1.hash_to_point(m).pair(pk[1])
True
```

Indeed, the pairing operator is bilinear. For example:

```
>>> a, b = 13, 29
>>> A = a * G1.generator()
>>> B = b * G2.generator()
>>> A.pair(B) == (a*b) * G1.generator().pair(G2.generator())
True
```

class petrelic.additive.pairing.**BilinearGroupPair**
A bilinear group pair.

Contains two origin groups G1, G2 and the image group GT.

groups()
Returns the three groups in the following order : G1, G2, GT.

class petrelic.additive.pairing.**G1**
G1 group.

classmethod generator()
Return generator of the group.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> generator + neutral == generator
True
```

classmethod hash_to_point(*hinput*)
Return group element obtained by hashing the input

Example:

```
>>> elem = G1.hash_to_point(b"foo")
>>> elem.is_valid()
True
```

classmethod infinity()
The point at infinity.

Alias for *G1.neutral_element()*

classmethod neutral_element()
Return the neutral element of the group G1.

In this case, the point at infinity.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> generator + neutral == generator
True
```

classmethod order()
Return the order of the group as a Bn large integer.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> order = G1.order()
>>> order * generator == neutral
True
```

classmethod sum(*elems*)
Efficient sum of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ x * G1.generator() for x in [10, 25, 13]]
>>> G1.sum(elems) ==  (10 + 25 + 13) * G1.generator()
True
```

classmethod wsum(*weights*, *elems*)
Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ x * G1.generator() for x in [10, 25, 13]]
>>> G1.wsum(weights, elems) ==  (1 * 10 + 2 * 25 + 3 * 13) * G1.
↪generator()
True
```

class petrelic.additive.pairing.**G1Element**
Element of the G1 group.

add(*other*)
Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 40 * G1.generator()
>>> a + b == 50 * G1.generator()
True
>>> a.add(b) == 50 * G1.generator()
True
```

double()
Return double of the current element

Example:

```
>>> generator = G1.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

eq(*other*)
Check point equality.

classmethod from_binary(*sbin*)
Deserialize a binary representation of the element of G1.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

get_affine_coordinates()
Return the affine coordinates (x,y) of this EC Point.

Example:

```
>>> generator = G1.generator()
>>> x, y = generator.get_affine_coordinates()
>>> x
Bn(3685416753713387016781088315183077757961620795782546409894578378688607592378376318883
>>> y
Bn(1339506544944476473020471379941921221584933875938349620426543736416511423956333506472
```

group
alias of *G1*

iadd (*other*)
Inplace add another point.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 10 * G1.generator()
>>> a += 3 * G1.generator()
>>> _ = b.iadd(3 * G1.generator())
>>> a == b
True
>>> a == 13 * G1.generator()
True
```

idouble ()
Inplace double the current element.

Example:

```
>>> generator = G1.generator()
>>> elem = G1.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

iinverse ()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G1.generator()
>>> elem2 = a * G1.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul (*other*)
Inplace point multiplication by a scalar

Examples:

```
>>> a = G1.generator()
>>> b = G1.generator()
>>> a *= 10
>>> _ = b.imul(10)
>>> a == b
True
```

(continues on next page)

(continued from previous page)

```
>>> a == 10 * G1.generator()
True
```

inverse()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G1.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G1.order() - a) * G1.generator()
True
```

is_infinity()
Check if the object is the neutral element of G1.

Example:

```
>>> generator = G1.generator()
>>> order = G1.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_neutral_element()
Check if the object is the neutral element of G1.

Example:

```
>>> generator = G1.generator()
>>> order = G1.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is a valid element on the curve. This method excludes the unity element. For that use *is_infinity*.

Example:

```
>>> elem = G1.hash_to_point(b"foo")
>>> elem.is_valid()
True
>>> elem = G1.infinity()
>>> elem.is_valid()
False
```

isub(*other*)
Inplace substract another point.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 10 * G1.generator()
>>> a -= 3 * G1.generator()
```

(continues on next page)

(continued from previous page)

```
>>> _ = b.isub(3 * G1.generator())
>>> a == b
True
>>> a == 7 * G1.generator()
True
```

mul (*other*)
Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = G1.generator()
>>> g + g == 2 * g
True
```

ne (*other*)
Check that the points are different.

neg ()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G1.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G1.order() - a) * G1.generator()
True
```

pair (*other*)
Pair element with another element in G2

Computes the bilinear pairing between self and another element in `petrelic.additive.pairing.G2`.

Examples:

```
>>> g1, g2 = G1.generator(), G2.generator()
>>> a, b = 10, 50
>>> A, B = g1 * a, g2 * b
>>> A.pair(B) == g1.pair(g2) * (a * b)
True
>>> A.pair(g2) == g1.pair(g2 * a)
True
>>> A.pair(g2) == g1.pair(g2) * a
True
```

sub (*other*)
Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * G1.generator()
>>> b = 13 * G1.generator()
```

(continues on next page)

(continued from previous page)

```
>>> a - b == 37 * G1.generator()
True
>>> a.sub(b) == 37 * G1.generator()
True
```

to_binary (*compressed=True*)
Serialize the element of G1 into a binary representation.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

class petrelic.additive.pairing.**G2**
G2 group.

classmethod generator ()
Return generator of the group.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> generator + neutral == generator
True
```

classmethod hash_to_point (*hinput*)
Return group element obtained by hashing the input

Example:

```
>>> elem = G2.hash_to_point(b"foo")
>>> elem.is_valid()
True
```

classmethod infinity ()
The point at infinity.

Alias for *G1.neutral_element()*

classmethod neutral_element ()
Return the neutral element of the group G2.

In this case, the point at infinity.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> generator + neutral == generator
True
```

classmethod order ()
Return the order of the EC group as a Bn large integer.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> order = G2.order()
>>> order * generator == neutral
True
```

classmethod sum(*elems*)

Efficient sum of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ x * G2.generator() for x in [10, 25, 13]]
>>> G2.sum(elems) ==  (10 + 25 + 13) * G2.generator()
True
```

classmethod wsum(*weights, elems*)

Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ x * G2.generator() for x in [10, 25, 13]]
>>> G2.wsum(weights, elems) ==  (1 * 10 + 2 * 25 + 3 * 13) * G2.
↪generator()
True
```

class petrelic.additive.pairing.**G2Element**

Element of the G2 group.

add(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 40 * G2.generator()
>>> a + b == 50 * G2.generator()
True
>>> a.add(b) == 50 * G2.generator()
True
```

double()

Return double of the current element

Example:

```
>>> generator = G2.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

eq(*other*)

Check that the points on the EC are equal.

classmethod from_binary(*sbin*)

Deserialize a binary representation of the element of G2.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

group

alias of *G2*

iadd(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 40 * G2.generator()
>>> a + b == 50 * G2.generator()
True
>>> a.add(b) == 50 * G2.generator()
True
```

idouble()

Inplace double the current element.

Example:

```
>>> generator = G2.generator()
>>> elem = G2.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

iinverse()

Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G2.generator()
>>> elem2 = a * G2.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)

Inplace point multiplication by a scalar

Examples:

```
>>> a = G2.generator()
>>> b = G2.generator()
>>> a *= 10
>>> _ = b.imul(10)
```

(continues on next page)

(continued from previous page)

```
>>> a == b
True
>>> a == 10 * G2.generator()
True
```

inverse()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G2.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G2.order() - a) * G2.generator()
True
```

is_infinity()
Check if the object is the neutral element of G2.

Example:

```
>>> generator = G2.generator()
>>> order = G2.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_neutral_element()
Check if the object is the neutral element of G2.

Example:

```
>>> generator = G2.generator()
>>> order = G2.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is a valid element on the curve. This method excludes the unity element. For that use *is_infinity*.

Example:

```
>>> elem = G2.hash_to_point(b"foo")
>>> elem.is_valid()
True
>>> elem = G2.infinity()
>>> elem.is_valid()
False
```

isub(*other*)
Inplace substract another point.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 10 * G2.generator()
>>> a -= 3 * G2.generator()
>>> _ = b.isub(3 * G2.generator())
>>> a == b
True
>>> a == 7 * G2.generator()
True
```

mul (*other*)

Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = G2.generator()
>>> g + g == 2 * g
True
```

ne (*other*)

Check that the points on the EC are not equal.

neg ()

Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G2.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G2.order() - a) * G2.generator()
True
```

sub (*other*)

Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * G2.generator()
>>> b = 13 * G2.generator()
>>> a - b == 37 * G2.generator()
True
>>> a.sub(b) == 37 * G2.generator()
True
```

to_binary (*compressed=True*)

Serialize the element of G2 into a binary representation.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

class petrelic.additive.pairing.**GT**
GT group.

classmethod generator()
Return generator of the group.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> generator + neutral == generator
True
```

classmethod infinity()
The unity element

Alias for *GT.neutral_element()*

classmethod neutral_element()
Return the neutral element of the group G1.

In this case, the point at infinity.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> generator + neutral == generator
True
```

classmethod order()
Return the order of the group as a Bn large integer.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> order = GT.order()
>>> order * generator == neutral
True
```

classmethod sum(*elems*)
Efficient sum of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ x * GT.generator() for x in [10, 25, 13]]
>>> GT.sum(elems) ==  (10 + 25 + 13) * GT.generator()
True
```

classmethod wsum(*weights*, *elems*)
Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ x * GT.generator() for x in [10, 25, 13]]
```

(continues on next page)

(continued from previous page)

```
>>> GT.wsum(weights, elems) ==  (1 * 10 + 2 * 25 + 3 * 13) * GT.
↪generator()
True
```

class petrelic.additive.pairing.**GTElement**

GT element.

add(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * GT.generator()
>>> b = 40 * GT.generator()
>>> a + b == 50 * GT.generator()
True
>>> a.add(b) == 50 * GT.generator()
True
```

double()

Return double of the current element

Example:

```
>>> generator = GT.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

eq(*other*)

Check that the points are equal.

classmethod from_binary(*sbin*)

Deserialize a binary representation of the element of GT.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

group

alias of *GT*

iadd(*other*)

Inplace add another point.

Examples:

```
>>> a = 10 * GT.generator()
>>> b = 10 * GT.generator()
>>> a += 3 * GT.generator()
>>> _ = b.iadd(3 * GT.generator())
>>> a == b
True
```

(continues on next page)

(continued from previous page)

```
>>> a == 13 * GT.generator()
True
```

idouble()

Inplace double the current element.

Example:

```
>>> generator = GT.generator()
>>> elem = GT.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

iinverse()

Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = GT.generator() ** a
>>> elem2 = GT.generator() ** a
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)

Inplace point multiplication by a scalar

Examples:

```
>>> a = GT.generator()
>>> b = GT.generator()
>>> a *= 10
>>> _ = b.imul(10)
>>> a == b
True
>>> a == 10 * GT.generator()
True
```

inverse()

Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = GT.generator() ** a
>>> elem.inverse() == GT.generator() ** (G1.order() - a)
True
```

is_neutral_element()

Check if the object is the neutral element of GT.

Example:

```
>>> generator = GT.generator()
>>> order = GT.order()
>>> elem = generator ** order
```

(continues on next page)

(continued from previous page)

```
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is in the group

Example:

```
>>> elem = GT.generator() ** 1337
>>> elem.is_valid()
True
```

isub(*other*)
Inplace substract another point.

Examples:

```
>>> a = 10 * GT.generator()
>>> b = 10 * GT.generator()
>>> a -= 3 * GT.generator()
>>> _ = b.isub(3 * GT.generator())
>>> a == b
True
>>> a == 7 * GT.generator()
True
```

mul(*other*)
Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = GT.generator()
>>> g + g == 2 * g
True
```

ne(*other*)
Check that the points on the EC are not equal.

neg()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = GT.generator() ** a
>>> elem.inverse() == GT.generator() ** (G1.order() - a)
True
```

sub(*other*)
Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * GT.generator()
>>> b = 13 * GT.generator()
>>> a - b == 37 * GT.generator()
True
>>> a.sub(b) == 37 * GT.generator()
True
```

to_binary(*compressed=True*)
Serialize the element of GT into a binary representation.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

1.3.4 Multiplicative interface

The multiplicative interface uses a multiplicative notation for all three groups G1, G2, and GT. This ensures that the notation is closest to that of most cryptography papers. You can use this interface by importing

```
from petrelic.multiplicative.pairing import G1, G2, GT
```

petrelic.multiplicative.pairing

This module provides a Python wrapper around RELIC's pairings using a multiplicative interface: operations in `petrelic.multiplicative.pairings.G1`, `petrelic.multiplicative.pairings.G2`, and `petrelic.multiplicative.pairings.GT` are all written multiplicatively.

Let's see how we can use this interface to implement the Boney-Lynn-Shacham signature scheme for type III pairings. First we generate a private key:

```
>>> sk = G1.order().random()
```

which is a random integer modulo the group order. Note that for this setting, all three groups have the same order. Next, we generate the corresponding public key:

```
>>> pk = (G1.generator() ** sk, G2.generator() ** sk)
```

(For security in the type III setting, the first component is a necessary part of the public key. It is not actually used in the scheme.) To sign a message *m* we first hash it to the curve G1 using *`G1.hash_to_point()`* and then raise it to the power of the signing key *sk* to obtain a signature:

```
>>> m = b"Some message"
>>> signature = G1.hash_to_point(m) ** sk
```

Finally, we can use the pairing operator to verify the signature:

```
>>> signature.pair(G2.generator()) == G1.hash_to_point(m).pair(pk[1])
True
```

Indeed, the pairing operator is bilinear. For example:

```
>>> a, b = 13, 29
>>> A = G1.generator() ** a
>>> B = G2.generator() ** b
>>> A.pair(B) == G1.generator().pair(G2.generator()) ** (a * b)
True
```

class petrelic.multiplicative.pairing.**BilinearGroupPair**

A bilinear group pair.

Contains two origin groups G1, G2 and the image group GT.

groups()

Returns the three groups in the following order : G1, G2, GT.

class petrelic.multiplicative.pairing.**G1**

G1 group.

classmethod generator()

Return generator of the group.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> generator * neutral == generator
True
```

classmethod hash_to_point(*hinput*)

Return group element obtained by hashing the input

Example:

```
>>> elem = G1.hash_to_point(b"foo")
>>> elem.is_valid()
True
```

classmethod neutral_element()

Return the neutral element of the group G1.

In this case, the point at infinity.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> generator * neutral == generator
True
```

classmethod order()

Return the order of the group as a Bn large integer.

Example:

```
>>> generator = G1.generator()
>>> neutral = G1.neutral_element()
>>> order = G1.order()
>>> generator ** order == neutral
True
```

classmethod prod(*elems*)

Efficient product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ G1.generator() ** x for x in [10, 25, 13]]
>>> G1.prod(elems) ==  G1.generator() ** (10 + 25 + 13)
True
```

classmethod unity()

The unity element

Alias for *G1.neutral_element()*

classmethod wprod(*weights*, *elems*)

Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ G1.generator() ** x for x in [10, 25, 13]]
>>> G1.wprod(weights, elems) ==  G1.generator() ** (1 * 10 + 2 * 25 + 3 *␣
↪13)
True
```

class petrelic.multiplicative.pairing.**G1Element**

Element of the G1 group.

div(*other*)

Divide two points

This method is aliased by *a / b* and *a // b*.

Examples:

```
>>> a = G1.generator() ** 50
>>> b = G1.generator() ** 13
>>> a / b == G1.generator() ** 37
True
>>> a // b == G1.generator() ** 37
True
>>> a.div(b) == G1.generator() ** 37
True
```

eq(*other*)

Check point equality.

classmethod from_binary(*sbin*)

Deserialize a binary representation of the element of G1.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

get_affine_coordinates()
Return the affine coordinates (x,y) of this EC Point.

Example:

```
>>> generator = G1.generator()
>>> x, y = generator.get_affine_coordinates()
>>> x
Bn(3685416753713387016781088315183077757961620795782546409894578378688607592378376318836
>>> y
Bn(1339506544944476473020471379941921221584933875938349620426543736416511423956333506472
```

group
alias of *G1*

idiv(*other*)
Inplace division by another point

Examples:

```
>>> a = G1.generator() ** 10
>>> b = G1.generator() ** 10
>>> a /= G1.generator() ** 3
>>> _ = b.idiv(G1.generator() ** 3)
>>> a == b
True
>>> a == G1.generator() ** 7
True
```

iinverse()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G1.generator()
>>> elem2 = a * G1.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)
Inplace multiplication by another element

Examples:

```
>>> a = G1.generator() ** 10
>>> b = G1.generator() ** 10
>>> a *= G1.generator() ** 3
>>> _ = b.imul(G1.generator() ** 3)
>>> a == b
True
>>> a == G1.generator() ** 13
True
```

inverse()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G1.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G1.order() - a) * G1.generator()
True
```

ipow(*other*)

Inplace raise element to the power of a scalar

Examples:

```
>>> g = G1.generator()
>>> a = G1.generator()
>>> _ = a.ipow(3)
>>> g * g * g == a
True
```

is_neutral_element()

Check if the object is the neutral element of G1.

Example:

```
>>> generator = G1.generator()
>>> order = G1.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_valid()

Check if the element is a valid element on the curve. This method excludes the unity element. For that use *is_infinity*.

Example:

```
>>> elem = G1.hash_to_point(b"foo")
>>> elem.is_valid()
True
>>> elem = G1.infinity()
>>> elem.is_valid()
False
```

isquare()

Inplace square of the current element.

Example:

```
>>> elem = G1.generator()
>>> _ = elem.isquare()
>>> elem == G1.generator() ** 2
True
```

mul(*other*)

Multiply two elements

This method is aliased by *a * b*.

Examples:

```
>>> a = G1.generator() ** 10
>>> b = G1.generator() ** 40
>>> a * b == G1.generator() ** 50
True
>>> a.mul(b) == G1.generator() ** 50
True
```

ne (*other*)
Check that the points are different.

pair (*other*)
Pair element with another element in G2

Computes the bilinear pairing between self and another element in `petrelic.multiplicative.pairing.G2`.

Examples:

```
>>> g1, g2 = G1.generator(), G2.generator()
>>> a, b = 10, 50
>>> A, B = g1 ** a, g2 ** b
>>> A.pair(B) == g1.pair(g2) ** (a * b)
True
>>> A.pair(g2) == g1.pair(g2 ** a)
True
>>> A.pair(g2) == g1.pair(g2) ** a
True
```

pow (*other*)
Raise element to the power of a scalar

This method is aliased by *el ** n*.

Examples:

```
>>> g = G1.generator()
>>> g * g == g ** 2
True
>>> g * g == g.pow(2)
True
```

square ()
Return the square of the current element

Example:

```
>>> generator = G1.generator()
>>> elem = generator.square()
>>> elem == generator ** 2
True
```

to_binary (*compressed=True*)
Serialize the element of G1 into a binary representation.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
```

(continues on next page)

(continued from previous page)

```
>>> generator == elem
True
```

class petrelic.multiplicative.pairing.**G2**

classmethod generator()

Return generator of the group.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> generator + neutral == generator
True
```

classmethod hash_to_point(*hinput*)

Return group element obtained by hashing the input

Example:

```
>>> elem = G2.hash_to_point(b"foo")
>>> elem.is_valid()
True
```

classmethod neutral_element()

Return the neutral element of the group G2.

In this case, the point at infinity.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> generator + neutral == generator
True
```

classmethod order()

Return the order of the EC group as a Bn large integer.

Example:

```
>>> generator = G2.generator()
>>> neutral = G2.neutral_element()
>>> order = G2.order()
>>> order * generator == neutral
True
```

classmethod prod(*elems*)

Efficient product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ G2.generator() ** x for x in [10, 25, 13]]
>>> G2.prod(elems) ==  G2.generator() ** (10 + 25 + 13)
True
```

classmethod unity()
The unity element

Alias for *G2.neutral_element()*

classmethod wprod(*weights*, *elems*)
Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ G2.generator() ** x for x in [10, 25, 13]]
>>> G2.wprod(weights, elems) ==  G2.generator() ** (1 * 10 + 2 * 25 + 3 *␣
↪13)
True
```

class petrelic.multiplicative.pairing.**G2Element**
Element of the G2 group.

div(*other*)
Divide two points

This method is aliased by *a / b* and *a // b*.

Examples:

```
>>> a = G2.generator() ** 50
>>> b = G2.generator() ** 13
>>> a / b == G2.generator() ** 37
True
>>> a // b == G2.generator() ** 37
True
>>> a.div(b) == G2.generator() ** 37
True
```

eq(*other*)
Check that the points on the EC are equal.

classmethod from_binary(*sbin*)
Deserialize a binary representation of the element of G2.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

group
alias of *G2*

idiv(*other*)
Inplace division by another point

Examples:

```
>>> a = G2.generator() ** 10
>>> b = G2.generator() ** 10
```

(continues on next page)

(continued from previous page)

```
>>> a /= G2.generator() ** 3
>>> _ = b.idiv(G2.generator() ** 3)
>>> a == b
True
>>> a == G2.generator() ** 7
True
```

iinverse()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G2.generator()
>>> elem2 = a * G2.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)
Inplace multiplication by another element

Examples:

```
>>> a = G2.generator() ** 10
>>> b = G2.generator() ** 10
>>> a *= G2.generator() ** 3
>>> _ = b.imul(G2.generator() ** 3)
>>> a == b
True
>>> a == G2.generator() ** 13
True
```

inverse()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G2.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G2.order() - a) * G2.generator()
True
```

ipow(*other*)
Inplace raise element to the power of a scalar

Examples:

```
>>> g = G2.generator()
>>> a = G2.generator()
>>> _ = a.ipow(3)
>>> g * g * g == a
True
```

is_neutral_element()
Check if the object is the neutral element of G2.

Example:

```
>>> generator = G2.generator()
>>> order = G2.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is a valid element on the curve. This method excludes the unity element. For that use *is_infinity*.

Example:

```
>>> elem = G2.hash_to_point(b"foo")
>>> elem.is_valid()
True
>>> elem = G2.infinity()
>>> elem.is_valid()
False
```

isquare()
Inplace square of the current element.

Example:

```
>>> elem = G2.generator()
>>> _ = elem.isquare()
>>> elem == G2.generator() ** 2
True
```

mul(*other*)
Multiply two elements

This method is aliased by *a * b*.

Examples:

```
>>> a = G2.generator() ** 10
>>> b = G2.generator() ** 40
>>> a * b == G2.generator() ** 50
True
>>> a.mul(b) == G2.generator() ** 50
True
```

ne(*other*)
Check that the points on the EC are not equal.

pow(*other*)
Raise element to the power of a scalar

This method is aliased by *el ** n*.

Examples:

```
>>> g = G2.generator()
>>> g * g == g ** 2
True
>>> g * g == g.pow(2)
True
```

square()
Return the square of the current element

Example:

```
>>> generator = G2.generator()
>>> elem = generator.square()
>>> elem == generator ** 2
True
```

to_binary(*compressed=True*)
Serialize the element of G2 into a binary representation.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

class petrelic.multiplicative.pairing.**GT**
GT group.

classmethod generator()
Return generator of the EC group.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> generator * neutral == generator
True
```

classmethod neutral_element()
Return the neutral element of the group GT. In this case, the unity point.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> generator * neutral == generator
True
```

classmethod order()
Return the order of the EC group as a Bn large integer.

Example:

```
>>> generator = GT.generator()
>>> neutral = GT.neutral_element()
>>> order = GT.order()
>>> generator ** order == neutral
True
```

classmethod prod(*elems*)
Efficient product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> elems = [ GT.generator() ** x for x in [10, 25, 13]]
>>> GT.prod(elems) ==  GT.generator() ** (10 + 25 + 13)
True
```

classmethod unity()

The unity elements

Alias for *GT.neutral_element()*

classmethod wprod(*weights*, *elems*)

Efficient weighted product of a number of elements

In the current implementation this function is not optimized.

Example:

```
>>> weights = [1, 2, 3]
>>> elems = [ GT.generator() ** x for x in [10, 25, 13]]
>>> GT.wprod(weights, elems) ==  GT.generator() ** (1 * 10 + 2 * 25 + 3 *␣
↪13)
True
```

class petrelic.multiplicative.pairing.**GTElement**

GT element.

div(*other*)

Divide two points

This method is aliased by *a / b* and *a // b*.

Examples:

```
>>> a = GT.generator() ** 50
>>> b = GT.generator() ** 13
>>> a / b == GT.generator() ** 37
True
>>> a // b == GT.generator() ** 37
True
>>> a.div(b) == GT.generator() ** 37
True
```

eq(*other*)

Check that the points are equal.

classmethod from_binary(*sbin*)

Deserialize a binary representation of the element of GT.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

group

alias of *GT*

idiv(*other*)

Inplace division by another point

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 10
>>> a /= GT.generator() ** 3
>>> _ = b.idiv(GT.generator() ** 3)
>>> a == b
True
>>> a == GT.generator() ** 7
True
```

iinverse()

Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = GT.generator() ** a
>>> elem2 = GT.generator() ** a
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

imul(*other*)

Inplace multiplication by another element

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 10
>>> a *= GT.generator() ** 3
>>> _ = b.imul(GT.generator() ** 3)
>>> a == b
True
>>> a == GT.generator() ** 13
True
```

inverse()

Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = GT.generator() ** a
>>> elem.inverse() == GT.generator() ** (G1.order() - a)
True
```

ipow(*other*)

Inplace raise element to the power of a scalar

Examples:

```
>>> g = GT.generator()
>>> a = GT.generator()
>>> _ = a.ipow(3)
>>> g * g * g == a
True
```

is_neutral_element()

Check if the object is the neutral element of GT.

Example:

```
>>> generator = GT.generator()
>>> order = GT.order()
>>> elem = generator ** order
>>> elem.is_neutral_element()
True
```

is_unity()
Check if the object is the neutral element of GT.

Example:

```
>>> generator = GT.generator()
>>> order = GT.order()
>>> elem = generator ** order
>>> elem.is_neutral_element()
True
```

is_valid()
Check if the element is in the group

Example:

```
>>> elem = GT.generator() ** 1337
>>> elem.is_valid()
True
```

isquare()
Inplace square of the current element.

Example:

```
>>> elem = GT.generator()
>>> _ = elem.isquare()
>>> elem == GT.generator() ** 2
True
```

mul(*other*)
Multiply two elements

This method is aliased by *a * b*.

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 40
>>> a * b == GT.generator() ** 50
True
>>> a.mul(b) == GT.generator() ** 50
True
```

ne(*other*)
Check that the points on the EC are not equal.

pow(*other*)
Raise element to the power of a scalar

This method is aliased by *el ** n*.

Examples:

```
>>> g = GT.generator()
>>> g * g == g ** 2
True
>>> g * g == g.pow(2)
True
```

square()
Return the square of the current element

Example:

```
>>> generator = GT.generator()
>>> elem = generator.square()
>>> elem == generator ** 2
True
```

to_binary(*compressed=True*)
Serialize the element of GT into a binary representation.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

1.3.5 petrelic.petlib package

Submodules

petrelic.petlib.pairing module

This module provides a Python wrapper around RELIC's pairings using petlib's interface: operations in `petrelic.native.pairings.G1` and `petrelic.native.pairings.G2` are written additively, whereas operations in `petrelic.native.pairings.GT` are written multiplicatively.

class petrelic.petlib.pairing.**BilinearGroupPair**
Bases: `object`

A bilinear group pair.

Contains two origin groups G1, G2 and the image group GT. The underlying `bplib.bp.BpGroup` object is also embedded.

groups()
Returns the three groups in the following order : G1, G2, GT.

class petrelic.petlib.pairing.**G1Elem**
Bases: *petrelic.native.pairing.G1Element*

Element of the G1 group

export(*compressed=True*)
Serialize the element of G1 into a binary representation.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

classmethod from_binary(*sbin*, *group=None*)

Deserialize a binary representation of the element of G1.

Example:

```
>>> generator = G1.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G1Element.from_binary(bin_repr)
>>> generator == elem
True
```

get_affine()

Return the affine coordinates (x,y) of this EC Point.

Example:

```
>>> generator = G1.generator()
>>> x, y = generator.get_affine_coordinates()
>>> x
Bn(368541675371338701678108831518307775796162079578254640989457837868860759237837631883
>>> y
Bn(133950654494447647302047137994192122158493387593834962042654373641651142395633350647
```

group

alias of *G1Group*

is_infinite()

Check if the object is the neutral element of G1.

Example:

```
>>> generator = G1.generator()
>>> order = G1.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

pair(*other*)

Computes bilinear pairing with self and otherwise

Examples:

```
>>> G1 = G1Group()
>>> G2 = G2Group()
>>> GT = GTGroup()
>>> G1.generator().pair(G2.generator()) == GT.generator()
True
```

```
>>> p = 100 * G1.generator()
>>> q = 200 * G2.generator()
>>> p.pair(q) == GT.generator() ** 20000
True
```

pt_add(*other*)

Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 40 * G1.generator()
>>> a + b == 50 * G1.generator()
True
>>> a.add(b) == 50 * G1.generator()
True
```

pt_add_inplace(*other*)

Inplace add another point.

Examples:

```
>>> a = 10 * G1.generator()
>>> b = 10 * G1.generator()
>>> a += 3 * G1.generator()
>>> _ = b.iadd(3 * G1.generator())
>>> a == b
True
>>> a == 13 * G1.generator()
True
```

pt_double()

Return double of the current element

Example:

```
>>> generator = G1.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

pt_double_inplace()

Inplace double the current element.

Example:

```
>>> generator = G1.generator()
>>> elem = G1.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

pt_eq(*other*)

Check point equality.

pt_mul(*other*)

Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = G1.generator()
>>> g + g == 2 * g
True
```

pt_mul_inplace(*other*)

Inplace point multiplication by a scalar

Examples:

```
>>> a = G1.generator()
>>> b = G1.generator()
>>> a *= 10
>>> _ = b.imul(10)
>>> a == b
True
>>> a == 10 * G1.generator()
True
```

pt_neg()

Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G1.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G1.order() - a) * G1.generator()
True
```

pt_neg_inplace()

Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G1.generator()
>>> elem2 = a * G1.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

pt_sub(*other*)

Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * G1.generator()
>>> b = 13 * G1.generator()
>>> a - b == 37 * G1.generator()
True
>>> a.sub(b) == 37 * G1.generator()
True
```

class petrelic.petlib.pairing.**G1Group**

Bases: *petrelic.native.pairing.G1*

G1 group

check_point (*pt*)
Ensures the point is on the curve.

Example:

```
>>> G = G1Group()
>>> G.check_point(G.generator())
True
>>> G.check_point(G.infinite())
True
```

classmethod infinite ()
The point at infinity.

Alias for `G1.neutral_element()`

class petrelic.petlib.pairing.**G2Elem**
Bases: *petrelic.native.pairing.G2Element*

Element of the G2 group

export (*compressed=True*)
Serialize the element of G2 into a binary representation.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

classmethod from_binary (*sbin*, *group=None*)
Deserialize a binary representation of the element of G2.

Example:

```
>>> generator = G2.generator()
>>> bin_repr = generator.to_binary()
>>> elem = G2Element.from_binary(bin_repr)
>>> generator == elem
True
```

group
alias of *G1Group*

is_infinite ()
Check if the object is the neutral element of G2.

Example:

```
>>> generator = G2.generator()
>>> order = G2.order()
>>> elem = order * generator
>>> elem.is_neutral_element()
True
```

pt_add (*other*)
Add two points together.

This method is aliased by *a* + *b*.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 40 * G2.generator()
>>> a + b == 50 * G2.generator()
True
>>> a.add(b) == 50 * G2.generator()
True
```

pt_add_inplace(*other*)
Add two points together.

This method is aliased by *a + b*.

Examples:

```
>>> a = 10 * G2.generator()
>>> b = 40 * G2.generator()
>>> a + b == 50 * G2.generator()
True
>>> a.add(b) == 50 * G2.generator()
True
```

pt_double()
Return double of the current element

Example:

```
>>> generator = G2.generator()
>>> elem = generator.double()
>>> elem == 2 * generator
True
```

pt_double_inplace()
Inplace double the current element.

Example:

```
>>> generator = G2.generator()
>>> elem = G2.generator()
>>> _ = elem.idouble()
>>> elem == 2 * generator
True
```

pt_eq(*other*)
Check that the points on the EC are equal.

pt_mul(*other*)
Multiply point by a scalar

This method is aliased by *n * pt*.

Examples:

```
>>> g = G2.generator()
>>> g + g == 2 * g
True
```

pt_mul_inplace(*other*)
Inplace point multiplication by a scalar

Examples:

```
>>> a = G2.generator()
>>> b = G2.generator()
>>> a *= 10
>>> _ = b.imul(10)
>>> a == b
True
>>> a == 10 * G2.generator()
True
```

pt_neg()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = a * G2.generator()
>>> -elem == elem.inverse()
True
>>> elem.inverse() == (G2.order() - a) * G2.generator()
True
```

pt_neg_inplace()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = a * G2.generator()
>>> elem2 = a * G2.generator()
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

pt_sub(*other*)
Substract two points

This method is aliased by *a - b*.

Examples:

```
>>> a = 50 * G2.generator()
>>> b = 13 * G2.generator()
>>> a - b == 37 * G2.generator()
True
>>> a.sub(b) == 37 * G2.generator()
True
```

class petrelic.petlib.pairing.**G2Group**
Bases: *petrelic.native.pairing.G2*

G2 group

check_point(*pt*)
Ensures the point is on the curve.

Example:

```
>>> G = G2Group()
>>> G.check_point(G.generator())
True
>>> G.check_point(G.infinite())
True
```

classmethod infinite()

The point at infinity.

Alias for `G2.neutral_element()`

class petrelic.petlib.pairing.**GTElem**

Bases: *petrelic.native.pairing.GTElement*

GT element

exp(*other*)

Raise element to the power of a scalar

This method is aliased by *el ** n*.

Examples:

```
>>> g = GT.generator()
>>> g * g == g ** 2
True
>>> g * g == g.pow(2)
True
```

exp_inplace(*other*)

Inplace raise element to the power of a scalar

Examples:

```
>>> g = GT.generator()
>>> a = GT.generator()
>>> _ = a.ipow(3)
>>> g * g * g == a
True
```

export(*compressed=True*)

Serialize the element of GT into a binary representation.

Example:

```
>>> generator = GT.generator()
>>> bin_repr = generator.to_binary()
>>> elem = GTElement.from_binary(bin_repr)
>>> generator == elem
True
```

classmethod from_binary(*sbin*, *group=None*)

Create an element from a byte sequence.

It accepts (but ignores) group as extra argument.

Example:

```
>>> G = GTGroup()
>>> byte_string = G.generator().export()                 # Export EC point␣
↪as byte string
```

(continues on next page)

(continued from previous page)

```
>>> GTElem.from_binary(byte_string, G) == G.generator()     # Import EC␣
↪point from binary string
True
>>> GTElem.from_binary(byte_string) == G.generator()     # Import EC point␣
↪from binary string
True
```

group
alias of *GTGroup*

inv()
Return the inverse of the element.

Examples:

```
>>> a = 30
>>> elem = GT.generator() ** a
>>> elem.inverse() == GT.generator() ** (G1.order() - a)
True
```

inv_inplace()
Inplace inverse of the current element

Examples:

```
>>> a = 30
>>> elem1 = GT.generator() ** a
>>> elem2 = GT.generator() ** a
>>> _ = elem1.iinverse()
>>> elem1 == elem2.inverse()
True
```

mul_inplace(*other*)
Inplace multiplication by another element

Examples:

```
>>> a = GT.generator() ** 10
>>> b = GT.generator() ** 10
>>> a *= GT.generator() ** 3
>>> _ = b.imul(GT.generator() ** 3)
>>> a == b
True
>>> a == GT.generator() ** 13
True
```

sqr()
Return the square of the current element

Example:

```
>>> generator = GT.generator()
>>> elem = generator.square()
>>> elem == generator ** 2
True
```

sqr_inplace()
Inplace square of the current element.

Example:

```
>>> elem = GT.generator()
>>> _ = elem.isquare()
>>> elem == GT.generator() ** 2
True
```

class petrelic.petlib.pairing.**GTGroup**

Bases: *petrelic.native.pairing.GT*

GT group

check_elem(*pt*)

Ensures the element is an element of the group

Example:

```
>>> G = GTGroup()
>>> G.check_elem(G.generator())
True
>>> G.check_elem(G.unity())
True
```

Module contents

Python Module Index

p

Index

A

B

C

D

E

F

G

M