Namespace chia.dotnet.bls

Classes

[AffinePoint](#)

Represents an affine point on an elliptic curve.

[AugSchemeMPL](#)

Provides extension methods for the AugSchemeMPL signature scheme. This is the scheme Chia uses for its signatures.

[ByteUtils](#)

Extension methods for working with byte arrays and conversions.

[Hmac](#)

Provides methods for HMAC (Hash-based Message Authentication Code) operations.

[JacobianPoint](#)

Represents a point in Jacobian coordinates on an elliptic curve. It can represent both a PublicKey and a Signature.

[KeyDerivation](#)

Provides methods for key derivation in the BLS cryptography library.

[PrivateKey](#)

Represents a private key used in BLS cryptography.

Enums

[Endian](#)

Enum representing the endianness of byte arrays.

Class AffinePoint

Namespace: [chia](#).[dotnet](#).[bls](#)

Assembly: chia-dotnet-bls.dll

Represents an affine point on an elliptic curve.

```
public class AffinePoint
```

Inheritance

[object](#) ← AffinePoint

Inherited Members

[object.Equals(object)](#) , [object.Equals(object, object)](#) , [object.GetHashCode()](#) , [object.GetType()](#) ,
[object.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

Properties

IsInfinity

Gets a value indicating whether the affine point is at infinity.

```
public bool IsInfinity { get; }
```

Property Value

[bool](#)

Represents an affine point on an elliptic curve.

IsOnCurve

Gets a value indicating whether the affine point is on the curve.

```
public bool IsOnCurve { get; }
```

Property Value

[bool⤢](#)

Represents an affine point on an elliptic curve.

Methods

Add(AffinePoint)

Adds another affine point to the current affine point.

```
public AffinePoint Add(AffinePoint value)
```

Parameters

`value` [AffinePoint](#)

The affine point to add.

Returns

[AffinePoint](#)

The sum of the two affine points.

Clone()

Creates a new instance that is a copy of the current affine point.

```
public AffinePoint Clone()
```

Returns

[AffinePoint](#)

A new instance that is a copy of the current affine point.

Double()

Doubles the affine point.

```
public AffinePoint Double()
```

Returns

AffinePoint

The doubled affine point.

Equals(AffinePoint)

Determines whether the specified object is equal to the current affine point.

```
public bool Equals(AffinePoint value)
```

Parameters

`value` AffinePoint

The object to compare with the current affine point.

Returns

bool↗

`true` if the specified object is equal to the current affine point; otherwise, `false`.

Multiply(BigInteger)

Multiplies the affine point by a scalar value.

```
public AffinePoint Multiply(BigInteger value)
```

Parameters

value BigInteger⧉

The scalar value.

Returns

[AffinePoint](#)

The result of the scalar multiplication.

Negate()

Negates the affine point.

```
public AffinePoint Negate()
```

Returns

[AffinePoint](#)

The negated affine point.

Subtract(AffinePoint)

Subtracts another affine point from the current affine point.

```
public AffinePoint Subtract(AffinePoint value)
```

Parameters

value [AffinePoint](#)

The affine point to subtract.

Returns

[AffinePoint](#)

The difference between the two affine points.

ToJacobian()

Converts the affine point to its Jacobian representation.

```
public JacobianPoint ToJacobian()
```

Returns

[JacobianPoint](#)

The Jacobian representation of the affine point.

ToString()

Returns a string that represents the current affine point.

```
public override string ToString()
```

Returns

[string](#)

A string that represents the current affine point.

Twist()

Twists the affine point.

```
public AffinePoint Twist()
```

Returns

[AffinePoint](#)

The twisted affine point.

Untwist()

Untwists the affine point.

```
public AffinePoint Untwist()
```

Returns

AffinePoint

The untwisted affine point.

Class AugSchemeMPL

Namespace: chia.dotnet.bls

Assembly: chia-dotnet-bls.dll

Provides extension methods for the AugSchemeMPL signature scheme. This is the scheme Chia uses for its signatures.

```
public static class AugSchemeMPL
```

Inheritance

object ← AugSchemeMPL

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Aggregate(JacobianPoint[])

Aggregates multiple signatures into a single signature.

```
public static JacobianPoint Aggregate(JacobianPoint[] signatures)
```

Parameters

`signatures` JacobianPoint[]

The array of signatures to be aggregated.

Returns

JacobianPoint

The aggregated signature as a JacobianPoint.

AggregateVerify(JacobianPoint[], byte[][], JacobianPoint)

Verifies an aggregated signature against multiple public keys and messages.

```
public static bool AggregateVerify(this JacobianPoint[] publicKeys, byte[][] messages,
JacobianPoint signature)
```

Parameters

`publicKeys` [JacobianPoint][]

The array of public keys used for verification.

`messages` [byte][][]

The array of messages to be verified.

`signature` [JacobianPoint]

The aggregated signature to be verified.

Returns

[bool]

True if the aggregated signature is valid, false otherwise.

DeriveChildPkUnhardened(JacobianPoint, long)

Derives a child unhardened public key from the specified public key and index.

```
public static JacobianPoint DeriveChildPkUnhardened(this JacobianPoint publicKey,
long index)
```

Parameters

`publicKey` [JacobianPoint]

The parent public key.

`index` [long]

The index of the child public key.

Returns

[JacobianPoint](#)

The derived child unhardened public key.

DeriveChildSk(PrivateKey, long)

Derives a child private key from the specified private key and index.

```
public static PrivateKey DeriveChildSk(this PrivateKey privateKey, long index)
```

Parameters

privateKey [PrivateKey](#)

The parent private key.

index [long](#)

The index of the child private key.

Returns

[PrivateKey](#)

The derived child private key.

DeriveChildSkUnhardened(PrivateKey, long)

Derives a child unhardened private key from the specified private key and index.

```
public static PrivateKey DeriveChildSkUnhardened(this PrivateKey privateKey, long index)
```

Parameters

privateKey [PrivateKey](#)

The parent private key.

index long⧉

    The index of the child private key.

Returns

[PrivateKey](#)

    The derived child unhardened private key.

KeyGen(byte[])

Generates a private key from the given seed.

```
public static PrivateKey KeyGen(this byte[] seed)
```

Parameters

seed byte⧉[]

    The seed used to generate the private key.

Returns

[PrivateKey](#)

    The generated private key.

Sign(PrivateKey, byte[])

Signs a message using the specified private key.

```
public static JacobianPoint Sign(this PrivateKey privateKey, byte[] message)
```

Parameters

privateKey [PrivateKey](#)

The private key used for signing.

message byte⧉[]

The message to be signed.

Returns

JacobianPoint

The signature as a JacobianPoint.

Sign(PrivateKey, string)

Signs a message using the specified private key.

```
public static JacobianPoint Sign(this PrivateKey privateKey, string message)
```

Parameters

privateKey PrivateKey

The private key used for signing.

message string⧉

The message to be signed.

Returns

JacobianPoint

The signature as a JacobianPoint.

Verify(JacobianPoint, byte[], JacobianPoint)

Verifies the signature of a message using the specified public key.

```
public static bool Verify(this JacobianPoint publicKey, byte[] message,
JacobianPoint signature)
```

Parameters

`publicKey` [JacobianPoint](#)

    The public key used for verification.

`message` [byte↗](#)[]

    The message to be verified.

`signature` [JacobianPoint](#)

    The signature to be verified.

Returns

[bool↗](#)

    True if the signature is valid, false otherwise.

Verify(JacobianPoint, string, JacobianPoint)

Verifies the signature of a message using the specified public key.

```
public static bool Verify(this JacobianPoint publicKey, string message,
JacobianPoint signature)
```

Parameters

`publicKey` [JacobianPoint](#)

    The public key used for verification.

`message` [string↗](#)

    The message to be verified.

`signature` [JacobianPoint](#)

    The signature to be verified.

Returns

[bool](#)⬈

True if the signature is valid, false otherwise.

Class ByteUtils

Namespace: [chia](#).[dotnet](#).[bls](#)

Assembly: chia-dotnet-bls.dll

Extension methods for working with byte arrays and conversions.

```
public static class ByteUtils
```

Inheritance

[object](#) ← ByteUtils

Inherited Members

[object.Equals(object)](#) , [object.Equals(object, object)](#) , [object.GetHashCode()](#) , [object.GetType()](#) ,
[object.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#) , [object.ToString()](#)

Methods

BigIntBitLength(BigInteger)

Calculates the number of bits required to represent a BigInteger.

```
public static long BigIntBitLength(BigInteger value)
```

Parameters

`value` [BigInteger](#)

   The BigInteger value.

Returns

[long](#)

   The number of bits required to represent the BigInteger.

BigIntToBits(BigInteger)

Converts a BigInteger to a byte array representing its binary representation.

```
public static byte[] BigIntToBits(this BigInteger i)
```

Parameters

`i` [BigInteger↗](#)

The BigInteger to convert.

Returns

[byte↗](#)[]

The byte array representing the binary representation of the BigInteger.

BigIntToBytes(BigInteger, int, Endian, bool)

Converts a BigInteger to a byte array.

```
public static byte[] BigIntToBytes(this BigInteger value, int size, Endian endian =
Endian.Big, bool signed = false)
```

Parameters

`value` [BigInteger↗](#)

The BigInteger to convert.

`size` [int↗](#)

The size of the resulting byte array.

`endian` [Endian](#)

The endianness of the byte array.

`signed` [bool↗](#)

Indicates whether the value is signed or unsigned.

Returns

byte □ []

    The byte array representation of the BigInteger.

BytesEqual(byte[], byte[])

Checks if two byte arrays are equal.

```
public static bool BytesEqual(this byte[] a, byte[] b)
```

Parameters

a byte □ []

    The first byte array.

b byte □ []

    The second byte array.

Returns

bool □

    True if the byte arrays are equal, false otherwise.

BytesToBigInt(byte[], Endian, bool)

Converts a byte array to a BigInteger.

```
public static BigInteger BytesToBigInt(this byte[] bytes, Endian endian = Endian.Big, bool
signed = false)
```

Parameters

bytes byte □ []

    The byte array to convert.

endian Endian

The endianness of the byte array.

signed bool⧉

Indicates whether the value is signed or unsigned.

Returns

BigInteger⧉

The BigInteger representation of the byte array.

BytesToInt(byte[], Endian, bool)

Converts a byte array to a long integer.

```
public static long BytesToInt(this byte[] bytes, Endian endian = Endian.Big, bool signed
= false)
```

Parameters

bytes byte⧉[]

The byte array to convert.

endian Endian

The endianness of the byte array.

signed bool⧉

Indicates whether the value is signed or unsigned.

Returns

long⧉

The long integer representation of the byte array.

ConcatenateArrays(params byte[][])

Concatenates multiple byte arrays into a single byte array.

```
public static byte[] ConcatenateArrays(params byte[][] arrays)
```

Parameters

`arrays` [byte⊕][][]

The byte arrays to concatenate.

Returns

[byte⊕][]

The concatenated byte array.

EncodeInt(long)

Converts a long integer to a byte array, as signed and big endian, with a special case for zero returning an empty array.

```
public static byte[] EncodeInt(this long value)
```

Parameters

`value` [long⊕]

The long integer to convert.

Returns

[byte⊕][]

The byte array representation of the long integer.

FromHex(string)

Converts a hexadecimal string to a byte array.

```
public static byte[] FromHex(this string hex)
```

Parameters

hex [string](#)↗

    The hexadecimal string to convert.

Returns

[byte](#)↗[]

    The byte array representation of the hexadecimal string.

HexStringToByteArray(string)

Converts a hexadecimal string to a byte array.

```
public static byte[] HexStringToByteArray(this string hex)
```

Parameters

hex [string](#)↗

    The hexadecimal string to convert.

Returns

[byte](#)↗[]

    The byte array representation of the hexadecimal string.

IntBitLength(long)

Calculates the number of bits required to represent a BigInteger.

```
public static int IntBitLength(long value)
```

Parameters

value long↗

   The BigInteger value.

Returns

int↗

   The number of bits required to represent the BigInteger.

IntToBytes(long, int, Endian, bool)

Converts a long integer to a byte array.

```
public static byte[] IntToBytes(this long value, int size, Endian endian = Endian.Big, bool
signed = false)
```

Parameters

value long↗

   The long integer to convert.

size int↗

   The size of the resulting byte array.

endian Endian

   The endianness of the byte array.

signed bool↗

   Indicates whether the value is signed or unsigned.

Returns

byte []

The byte array representation of the long integer.

ToBytes(string)

Converts a string to a byte array using UTF-8 encoding.

```
public static byte[] ToBytes(this string value)
```

Parameters

value string

The string to convert.

Returns

byte []

The byte array representation of the string.

ToHex(byte[])

Converts a byte array to a hexadecimal string.

```
public static string ToHex(this byte[] bytes)
```

Parameters

bytes byte []

The byte array to convert.

Returns

string

The hexadecimal string representation of the byte array.

Enum Endian

Namespace: [chia](.).[dotnet](.).[bls](.)

Assembly: chia-dotnet-bls.dll

Enum representing the endianness of byte arrays.

```
public enum Endian
```

Fields

Big = 1

Big endian byte order.

Little = 0

Little endian byte order.

Class Hmac

Namespace: [chia](.).[dotnet](.).[bls]

Assembly: chia-dotnet-bls.dll

Provides methods for HMAC (Hash-based Message Authentication Code) operations.

```
public static class Hmac
```

Inheritance

[object](#) ← Hmac

Inherited Members

[object.Equals(object)](#) , [object.Equals(object, object)](#) , [object.GetHashCode()](#) , [object.GetType()](#) , [object.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#) , [object.ToString()](#)

Fields

HmacBlockSize

The block size used in HMAC operations.

```
public const int HmacBlockSize = 64
```

Field Value

[int](#)

Provides methods for HMAC (Hash-based Message Authentication Code) operations.

Methods

Hash256(ArraySegment<byte>)

Computes the SHA-256 hash of the specified message segment.

```
public static byte[] Hash256(ArraySegment<byte> messageSegment)
```

Parameters

`messageSegment` [ArraySegment⧉](#) `<` [byte⧉](#) `>`

   The segment of the message to hash.

Returns

[byte⧉](#) []

   The SHA-256 hash of the message segment.

Hash256(byte[])

Computes the SHA-256 hash of the specified message.

```
public static byte[] Hash256(this byte[] message)
```

Parameters

`message` [byte⧉](#) []

   The message to hash.

Returns

[byte⧉](#) []

   The SHA-256 hash of the message.

Hash512(byte[])

Computes the SHA-512 hash of the specified message.

```
public static byte[] Hash512(this byte[] message)
```

Parameters

message byte⬀[]

The message to hash.

Returns

byte⬀[]

The SHA-512 hash of the message.

Hmac256(byte[], byte[])

Computes the HMAC-SHA-256 of the specified message using the specified key.

```
public static byte[] Hmac256(byte[] message, byte[] k)
```

Parameters

message byte⬀[]

The message to compute the HMAC for.

k byte⬀[]

The key to use for HMAC computation.

Returns

byte⬀[]

The HMAC-SHA-256 of the message.

Class JacobianPoint

Namespace: [chia](.).[dotnet](.).[bls](.)

Assembly: chia-dotnet-bls.dll

Represents a point in Jacobian coordinates on an elliptic curve. It can represent both a PublicKey and a Signature.

```
public class JacobianPoint
```

Inheritance

[object](.) ← JacobianPoint

Inherited Members

[object.Equals(object)](.) , [object.Equals(object, object)](.) , [object.GetHashCode()](.) , [object.GetType()](.) , [object.MemberwiseClone()](.) , [object.ReferenceEquals(object, object)](.)

Extension Methods

[AugSchemeMPL.DeriveChildPkUnhardened(JacobianPoint, long)](.) , [AugSchemeMPL.Verify(JacobianPoint, byte[], JacobianPoint)](.) , [AugSchemeMPL.Verify(JacobianPoint, string, JacobianPoint)](.) , [KeyDerivation.CalculateSyntheticOffset(JacobianPoint, byte[])](.) , [KeyDerivation.CalculateSyntheticPublicKey(JacobianPoint, byte[])](.) , [KeyDerivation.DerivePublicKeyPath(JacobianPoint, int[])](.) , [KeyDerivation.DerivePublicKeyWallet(JacobianPoint, int)](.)

Properties

IsInfinity

A flag indicating whether the point is at infinity.

```
public bool IsInfinity { get; }
```

Property Value

[bool](.)

Represents a point in Jacobian coordinates on an elliptic curve. It can represent both a PublicKey and a Signature.

Methods

Add(JacobianPoint)

Adds the point to another point.

```
public JacobianPoint Add(JacobianPoint value)
```

Parameters

`value` JacobianPoint

The other point

Returns

JacobianPoint

The resulting point

Clone()

Clones the point.

```
public JacobianPoint Clone()
```

Returns

JacobianPoint

The cloned point

Double()

Doubles the point.

```
public JacobianPoint Double()
```

Returns

[JacobianPoint](#)

   [JacobianPoint](#)

Equals(JacobianPoint)

Compares the point to another point.

```
public bool Equals(JacobianPoint value)
```

Parameters

`value` [JacobianPoint](#)

   The other point

Returns

[bool](#)⤴

   True if they are equal

FromBytes(byte[], bool)

Creates a JacobianPoint from a byte array.

```
public static JacobianPoint FromBytes(byte[] bytes, bool isExtension)
```

Parameters

`bytes` [byte](#)⤴[]

`isExtension` [bool](#)⤴

Returns

JacobianPoint

JacobianPoint

FromBytesG1(byte[], bool)

Creates a JacobianPoint from a byte array.

```
public static JacobianPoint FromBytesG1(byte[] bytes, bool isExtension = false)
```

Parameters

bytes byte[]

isExtension bool

Returns

JacobianPoint

FromBytesG2(byte[], bool)

Creates a JacobianPoint from a byte array.

```
public static JacobianPoint FromBytesG2(byte[] bytes, bool isExtension = true)
```

Parameters

bytes byte[]

isExtension bool

Returns

JacobianPoint

FromHex(string, bool)

Creates a JacobianPoint from a hexadecimal string.

```
public static JacobianPoint FromHex(string hex, bool isExtension)
```

Parameters

hex [string](#)

isExtension [bool](#)

Returns

[JacobianPoint](#)

FromHexG1(string, bool)

Creates a JacobianPoint from a hexadecimal string.

```
public static JacobianPoint FromHexG1(string hex, bool isExtension = false)
```

Parameters

hex [string](#)

isExtension [bool](#)

Returns

[JacobianPoint](#)

FromHexG2(string, bool)

Creates a JacobianPoint from a hexadecimal string.

```
public static JacobianPoint FromHexG2(string hex, bool isExtension = true)
```

Parameters

hex [string](#)

isExtension [bool](#)

Returns

[JacobianPoint](#)

GenerateG1()

Creates a JacobianPoint

```
public static JacobianPoint GenerateG1()
```

Returns

[JacobianPoint](#)

GenerateG2()

Creates a JacobianPoint

```
public static JacobianPoint GenerateG2()
```

Returns

[JacobianPoint](#)

GetFingerprint()

Gets the fingerprint of the point.

```
public long GetFingerprint()
```

Returns

[long](#)↗

InfinityG1(bool)

Creates a JacobianPoint at the G1 Infinity point.

```
public static JacobianPoint InfinityG1(bool isExtension = false)
```

Parameters

`isExtension` [bool](#)↗

Returns

[JacobianPoint](#)

InfinityG2(bool)

Creates a JacobianPoint at the G2 Infinity point.

```
public static JacobianPoint InfinityG2(bool isExtension = true)
```

Parameters

`isExtension` [bool](#)↗

Returns

[JacobianPoint](#)

IsOnCurve()

Checks if the point is on the curve.

```
public bool IsOnCurve()
```

Returns

[bool⍀](#)

IsValid()

Checks if the point is valid.`

```
public bool IsValid()
```

Returns

[bool⍀](#)

Multiply(BigInteger)

Multiplies the point by a scalar.

```
public JacobianPoint Multiply(BigInteger value)
```

Parameters

`value` [BigInteger⍀](#)

Returns

[JacobianPoint](#)

Negate()

Negates the point.

```
public JacobianPoint Negate()
```

Returns

JacobianPoint

    JacobianPoint

ToAffine()

Converts the point to an AffinePoint.

```
public AffinePoint ToAffine()
```

Returns

AffinePoint

    AffinePoint

ToBytes()

Converts the point to a byte array.

```
public byte[] ToBytes()
```

Returns

byte□ []

    The byte array

ToHex()

Converts the point to a hexadecimal string.

```
public string ToHex()
```

Returns

[string](#) ↗

    Hex string

ToString()

Converts the point to a string. This is an alias for [ToHex()](#).

```
public override string ToString()
```

Returns

[string](#) ↗

Class KeyDerivation

Namespace: [chia](#).[dotnet](#).[bls](#)

Assembly: chia-dotnet-bls.dll

Provides methods for key derivation in the BLS cryptography library.

```
public static class KeyDerivation
```

Inheritance

[object](#) ← KeyDerivation

Inherited Members
[object.Equals(object)](#) , [object.Equals(object, object)](#) , [object.GetHashCode()](#) , [object.GetType()](#) , [object.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#) , [object.ToString()](#)

Fields

DEFAULT_HIDDEN_PUZZLE_HASH

Default hidden puzzle hash used in key derivation.

```
public static readonly byte[] DEFAULT_HIDDEN_PUZZLE_HASH
```

Field Value

[byte](#) []

Provides methods for key derivation in the BLS cryptography library.

Methods

CalculateSyntheticOffset(JacobianPoint, byte[])

Calculates the synthetic offset for the given public key and hidden puzzle hash.

```
public static BigInteger CalculateSyntheticOffset(this JacobianPoint publicKey,
byte[] hiddenPuzzleHash)
```

Parameters

publicKey JacobianPoint

  The public key.

hiddenPuzzleHash byte[]

  The hidden puzzle hash.

Returns

BigInteger

  The synthetic offset.

CalculateSyntheticPrivateKey(PrivateKey, byte[])

Calculates the synthetic private key for the given private key and hidden puzzle hash.

```
public static PrivateKey CalculateSyntheticPrivateKey(this PrivateKey privateKey,
byte[] hiddenPuzzleHash)
```

Parameters

privateKey PrivateKey

  The original private key.

hiddenPuzzleHash byte[]

  The hidden puzzle hash.

Returns

PrivateKey

  The synthetic private key.

CalculateSyntheticPublicKey(JacobianPoint, byte[])

Calculates the synthetic public key for the given public key and hidden puzzle hash.

```
public static JacobianPoint CalculateSyntheticPublicKey(this JacobianPoint publicKey,
byte[] hiddenPuzzleHash)
```

Parameters

publicKey [JacobianPoint](#)

    The original public key.

hiddenPuzzleHash [byte](#)[]

    The hidden puzzle hash.

Returns

[JacobianPoint](#)

    The synthetic public key.

DerivePrivateKey(PrivateKey, int, bool)

Derives a private key from the given master private key.

```
public static PrivateKey DerivePrivateKey(this PrivateKey masterPrivateKey, int index,
bool hardened)
```

Parameters

masterPrivateKey [PrivateKey](#)

    The master private key.

index [int](#)

    The index of the derived key.

hardened [bool](#)

Indicates if the derivation should be hardened.

Returns

[PrivateKey](#)

The derived private key.

DerivePrivateKeyPath(PrivateKey, int[], bool)

Derives a private key path from the given master private key.

```
public static PrivateKey DerivePrivateKeyPath(this PrivateKey privateKey, int[] path,
bool hardened)
```

Parameters

`privateKey` [PrivateKey](#)

The master private key.

`path` [int](#)[]

The path to derive.

`hardened` [bool](#)

Indicates if the derivation should be hardened.

Returns

[PrivateKey](#)

The derived private key.

DerivePublicKeyPath(JacobianPoint, int[])

Derives a public key path from the given master public key.

```
public static JacobianPoint DerivePublicKeyPath(this JacobianPoint publicKey, int[] path)
```

Parameters

`publicKey` [JacobianPoint](#)

    The master public key.

`path` [int](#)[]

    The path to derive.

Returns

[JacobianPoint](#)

    The derived public key.

DerivePublicKeyWallet(JacobianPoint, int)

Derives a public key from the given master public key.

```
public static JacobianPoint DerivePublicKeyWallet(this JacobianPoint masterPublicKey,
int index)
```

Parameters

`masterPublicKey` [JacobianPoint](#)

    The master public key.

`index` [int](#)

    The index of the derived key.

Returns

[JacobianPoint](#)

    The derived public key.

Class PrivateKey

Namespace: [chia](#).[dotnet](#).[bls](#)

Assembly: chia-dotnet-bls.dll

Represents a private key used in BLS cryptography.

```
public class PrivateKey
```

Inheritance

[object](#) ← PrivateKey

Inherited Members

[object.Equals(object)](#) , [object.Equals(object, object)](#) , [object.GetHashCode()](#) , [object.GetType()](#) , [object.MemberwiseClone()](#) , [object.ReferenceEquals(object, object)](#)

Extension Methods

[AugSchemeMPL.DeriveChildSk(PrivateKey, long)](#) , [AugSchemeMPL.DeriveChildSkUnhardened(PrivateKey, long)](#) , [AugSchemeMPL.Sign(PrivateKey, byte[])](#) , [AugSchemeMPL.Sign(PrivateKey, string)](#) , [KeyDerivation.CalculateSyntheticPrivateKey(PrivateKey, byte[])](#) , [KeyDerivation.DerivePrivateKey(PrivateKey, int, bool)](#) , [KeyDerivation.DerivePrivateKeyPath(PrivateKey, int[], bool)](#)

Constructors

PrivateKey(BigInteger)

Initializes a new instance of the [PrivateKey](#) class with the specified value.

```
public PrivateKey(BigInteger value)
```

Parameters

`value` [BigInteger](#)

The value of the private key.

Fields

Length

The length of the private key in bytes.

```
public const byte Length = 48
```

Field Value

[byte⧉](#)

Represents a private key used in BLS cryptography.

Size

The size of the private key in bytes.

```
public const int Size = 32
```

Field Value

[int⧉](#)

Represents a private key used in BLS cryptography.

Value

The value of the private key as a BigInteger.

```
public readonly BigInteger Value
```

Field Value

[BigInteger⧉](#)

Represents a private key used in BLS cryptography.

Methods

Aggregate(PrivateKey[])

Aggregates an array of private keys into a single private key.

```
public static PrivateKey Aggregate(PrivateKey[] privateKeys)
```

Parameters

`privateKeys` [PrivateKey][]

    The array of private keys to aggregate.

Returns

[PrivateKey]

    The aggregated private key.

Equals(PrivateKey)

Determines whether the specified [PrivateKey] object is equal to the current [PrivateKey].

```
public bool Equals(PrivateKey value)
```

Parameters

`value` [PrivateKey]

    The [PrivateKey] object to compare with the current [PrivateKey].

Returns

[bool]↗

    `true` if the specified object is equal to the current object; otherwise, `false`.

FromBigInt(BigInteger)

Creates a [PrivateKey](#) instance from the specified BigInteger value.

```
public static PrivateKey FromBigInt(BigInteger value)
```

Parameters

`value` [BigInteger](#)⧉

    The BigInteger value representing the private key.

Returns

[PrivateKey](#)

    A new [PrivateKey](#) instance.

FromBytes(byte[])

Creates a [PrivateKey](#) instance from the specified byte array.

```
public static PrivateKey FromBytes(byte[] bytes)
```

Parameters

`bytes` [byte](#)⧉[]

    The byte array representing the private key.

Returns

[PrivateKey](#)

    A new [PrivateKey](#) instance.

FromHex(string)

Creates a [PrivateKey](#) instance from the specified hexadecimal string.

```
public static PrivateKey FromHex(string hex)
```

Parameters

hex [string](#)

The hexadecimal string representing the private key.

Returns

[PrivateKey](#)

A new [PrivateKey](#) instance.

FromSeed(byte[])

Creates a [PrivateKey](#) instance from the specified seed.

```
public static PrivateKey FromSeed(byte[] seed)
```

Parameters

seed [byte](#)[]

The seed used to generate the private key.

Returns

[PrivateKey](#)

A new [PrivateKey](#) instance.

FromSeed(string)

Creates a [PrivateKey](#) instance from the specified seed.

```
public static PrivateKey FromSeed(string seed)
```

Parameters

seed [string](#)⬈

    The seed used to generate the private key.

Returns

[PrivateKey](#)

    A new [PrivateKey](#) instance.

GetG1()

Gets the corresponding G1 point on the elliptic curve. The G1 point is the PublicKey.

```
public JacobianPoint GetG1()
```

Returns

[JacobianPoint](#)

    The G1 point.

ToBytes()

Converts the private key to a byte array.

```
public byte[] ToBytes()
```

Returns

[byte](#)⬈[]

    The byte array representation of the private key.

ToHex()

Converts the private key to a hexadecimal string.

```
public string ToHex()
```

Returns

[string](#)☐

The hexadecimal string representation of the private key.

ToString()

Returns a string that represents the current private key.

```
public override string ToString()
```

Returns

[string](#)☐

A string representation of the private key.