NAME

Crypt::OpenSSL::Bignum − OpenSSL's multiprecision integer arithmetic

SYNOPSIS

```
use Crypt::OpenSSL::Bignum;

my $bn = Crypt::OpenSSL::Bignum->new_from_decimal( "1000" );
# or
my $bn = Crypt::OpenSSL::Bignum->new_from_word( 1000 );
# or
my $bn = Crypt::OpenSSL::Bignum->new_from_hex("3e8"); # no leading 0x
# or
my $bn = Crypt::OpenSSL::Bignum->new_from_bin(pack( "C*", 3, 232 ))

use Crypt::OpenSSL::Bignum::CTX;

sub print_factorial
{
  my( $n ) = @_;
  my $fac = Crypt::OpenSSL::Bignum->one();
  my $ctx = Crypt::OpenSSL::Bignum::CTX->new();
  foreach my $i (1 .. $n)
  {
    $fac->mul( Crypt::OpenSSL::Bignum->new_from_word( $i ), $ctx, $fac );
  }
  print "$n factorial is ", $fac->to_decimal(), "\n";
}
```

DESCRIPTION

Crypt::OpenSSL::Bignum provides access to OpenSSL multiprecision integer arithmetic libraries. Presently, many though not all of the arithmetic operations that OpenSSL provides are exposed to perl. In addition, this module can be used to provide access to bignum values produced by other OpenSSL modules, such as key parameters from Crypt::OpenSSL::RSA.

NOTE: Many of the methods in this package can croak, so use eval, or Error.pm's try/catch mechanism to capture errors.

Constructors

new_from_decimal

```
my $bn = Crypt::OpenSSL::Bignum->new_from_decimal($decimal_string);
```

Create a new Crypt::OpenSSL::Bignum object whose value is specified by the given decimal representation.

new_from_hex

```
my $bn = Crypt::OpenSSL::Bignum->new_from_hex($hex_string); #no leading '0x'
```

Create a new Crypt::OpenSSL::Bignum object whose value is specified by the given hexadecimal representation.

new_from_word

```
my $bn = Crypt::OpenSSL::Bignum->new_from_word($unsigned_integer);
```

Create a new Crypt::OpenSSL::Bignum object whose value will be the word given. Note that numbers represented by objects created using this method are necessarily between 0 and 2^32 − 1.

new_from_bin

```
my $bn = Crypt::OpenSSL::Bignum->new_from_bin($bin_buffer);
```

Create a new Crypt::OpenSSL::Bignum object whose value is specified by the given packed binary string (created by ''to_bin''). Note that objects created using this method are necessarily nonnegative.

new

```
my $bn = Crypt::OpenSSL::Bignum->new;
```

Returns a new Crypt::OpenSSL::Bignum object representing 0

zero

```
my $bn = Crypt::OpenSSL::Bignum->zero;
```

Returns a new Crypt::OpenSSL::Bignum object representing 0 (same as new)

one

```
my $bn = Crypt::OpenSSL::Bignum->one;
```

Returns a new Crypt::OpenSSL::Bignum object representing 1

rand

```
my $bn = Crypt::OpenSSL::Bignum->rand($bits, $top, $bottom)
# $bits, $top, $bottom are integers
```

generates a cryptographically strong pseudo-random number of bits bits in length and stores it in rnd. If top is −1, the most significant bit of the random number can be zero. If top is 0, it is set to 1, and if top is 1, the two most significant bits of the number will be set to 1, so that the product of two such random numbers will always have 2*bits length. If bottom is true, the number will be odd.

pseudo_rand

```
my $bn = Crypt::OpenSSL::Bignum->pseudo_rand($bits, $top, $bottom)
# $bits, $top, $bottom are integers
```

does the same, but pseudo-random numbers generated by this function are not necessarily unpredictable. They can be used for non-cryptographic purposes and for certain purposes in cryptographic protocols, but usually not for key generation etc.

rand_range

```
my $bn = Crypt::OpenSSL::Bignum->rand_range($bn_range)
```

generates a cryptographically strong pseudo-random number rnd in the range 0 <lt>= rnd < range. **BN_pseudo_rand_range()** does the same, but is based on **BN_pseudo_rand()**, and hence numbers generated by it are not necessarily unpredictable.

bless_pointer

```
my $bn = Crypt::OpenSSL::Bignum->bless_pointer($BIGNUM_ptr)
```

Given a pointer to a OpenSSL BIGNUM object in memory, construct and return Crypt::OpenSSL::Bignum object around this. Note that the underlying BIGNUM object will be destroyed (via **BN_clear_free** (3ssl)) when the returned Crypt::OpenSSL::Bignum object is no longer referenced, so the pointer passed to this method should only be referenced via the returned perl object after calling bless_pointer.

This method is intended only for use by XSUB writers writing code that interfaces with OpenSSL library methods, and who wish to be able to return a BIGNUM structure to perl as a Crypt::OpenSSL::Bignum object.

Instance Methods

to_decimal

```
my $decimal_string = $self->to_decimal;
```

Return a decimal string representation of this object.

to_hex

```
my $hex_string = $self->to_hex;
```

Return a hexadecimal string representation of this object.

to_bin

```
my $bin_buffer = $self->to_bin;
```

Return a packed binary string representation of this object. Note that sign is ignored, so that to bin called on a Crypt::OpenSSL::Bignum object representing a negative number returns the same value as it would called on an object representing that number's absolute value.

get_word

```
my $unsigned_int = $self->get_word;
```

Return a scalar integer representation of this object, if it can be represented as an unsigned long.

is_zero

```
my $bool = $self->is_zero;
```

Returns true of this object represents 0.

is_one

```
my $bool = $self->is_one;
```

Returns true of this object represents 1.

is_odd

```
my $bool = $self->is_odd;
```

Returns true of this object represents an odd number.

add

```
my $new_bn_object = $self->add($bn_b); # $new_bn_object = $self + $bn_b
# or
$self->add($bn_b, $result_bn);         # $result_bn = $self + $bn_b
```

This method returns the sum of this object and the first argument. If only one argument is passed, a new Crypt::OpenSSL::Bignum object is created for the return value; otherwise, the value of second argument is set to the result and returned.

sub

```
my $new_bn_object = $self->sub($bn_b); # $new_bn_object = $self - $bn_b
# or
$self->sub($bn_b, $result_bn);         # $result_bn = $self - $bn_b
```

This method returns the difference of this object and the first argument. If only one argument is passed, a new Crypt::OpenSSL::Bignum object is created for the return value; otherwise, the value of second argument is set to the result and returned.

mul

```
my $new_bn_object = $self->mul($bn_b, $ctx); # $new_bn_object = $self * $bn_b
# or
$self->mul($bn_b, $ctx, $result_bn);         # $result_bn = $self * $bn_b
```

This method returns the product of this object and the first argument, using the second argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad. If only two arguments are passed, a new Crypt::OpenSSL::Bignum object is created for the return value; otherwise, the value of third argument is set to the result and returned.

div

```
my ($quotient, $remainder) = $self->div($bn_b, $ctx);
# or
$self->div($bn_b, $ctx, $quotient, $remainder);
```

This method returns a list consisting of quotient and the remainder obtained by dividing this object by the first argument, using the second argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad. If only two arguments are passed, new Crypt::OpenSSL::Bignum objects are created for both return values. If a third argument is passed, otherwise, the value of third argument is set to the

quotient. If a fourth argument is passed, the value of the fourth argument is set to the remainder.

mod

```
my $remainder = $self->mod($bn_b, $ctx);
# or
$self->mod($bn_b, $ctx, $remainder);
```

This method returns the remainder obtained by dividing this object by the first argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad. Crypt::OpenSSL::Bignum object is created for the return value. If a third argument is passed, the value of third argument is set to the remainder.

sqr

```
my $new_bn_object = $self->sqr($ctx);
# new object is created $self is not modified
```

This method returns the square (`$self ** 2`) of Crypt::OpenSSL::Bignum object.

exp

```
my $new_bn_object = $self->exp($bn_exp, $ctx);
# new object is created $self is not modified
```

This method returns the product of this object exponentiated by the first argument (Crypt::OpenSSL::Bignum object), using the second argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad.

mod_exp

```
my $new_bn_object = $self->exp_mod($bn_exp, $bn_mod, $ctx);
# new object is created $self is not modified
```

This method returns the product of this object exponentiated by the first argument (Crypt::OpenSSL::Bignum object), modulo the second argument (also Crypt::OpenSSL::Bignum object), using the third argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad.

mod_mul

```
my $new_bn_object = $self->mod_mul($bn_b, $bn_mod, $ctx);
# new object is created $self is not modified
```

This method returns `($self * $bn_b) % $bn_mod`, using the third argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad.

mod_inverse

```
my $new_bn_object = $self->mod_inverse($bn_n, $ctx);
# new object is created $self is not modified
```

Computes the inverse of `$self` modulo `$bn_n` and returns the result in a new Crypt::OpenSSL::Bignum object, using the second argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad.

gcd

```
my $new_bn_object = $self->gcd($bn_b, $ctx);
# new object is created $self is not modified
```

Computes the greatest common divisor of `$self` and `$bn_b` and returns the result in a new Crypt::OpenSSL::Bignum object, using the second argument, a Crypt::OpenSSL::Bignum::CTX object, as a scratchpad.

cmp

```
my $result = $self->cmp($bn_b);
#returns:
# -1 if self <  bn_b
#  0 if self == bn_b
#  1 if self >  bn_b
```

Comparison of values `$self` and `$bn_b` (Crypt::OpenSSL::Bignum objects).

ucmp

```
my $result = $self->ucmp($bn_b);
#returns:
# -1 if |self| <  |bn_b|
#  0 if |self| == |bn_b|
#  1 if |self| >  |bn_b|
```

Comparison using the absolute values of `$self` and `$bn_b` (Crypt::OpenSSL::Bignum objects).

equals

```
my $result = $self->equals($bn_b);
#returns:
# 1 if self == bn_b
# 0 otherwise
```

num_bits

```
my $bits = $self->num_bits;
```

Returns the number of significant bits in a word. If we take 0x00000432 as an example, it returns 11, not 16, not 32. Basically, except for a zero, it returns `floor(log2(w)) + 1`.

num_bytes

```
my $bytes = $self->num_bytes;
```

Returns the size of binary represenatation in bytes.

rshift

```
my $new_bn_object = $self->rshift($n);
# new object is created $self is not modified
```

Shifts a right by `$n` (integer) bits and places the result into a newly created Crypt::OpenSSL::Bignum object.

lshift

```
my $new_bn_object = $self->lshift($n);
# new object is created $self is not modified
```

Shifts a left by `$n` (integer) bits and places the result into a newly created Crypt::OpenSSL::Bignum object.

swap

```
my $bn_a = Crypt::OpenSSL::Bignum->new_from_decimal("1234567890001");
my $bn_b = Crypt::OpenSSL::Bignum->new_from_decimal("1234567890002");

$bn_a->swap($bn_b);
# or
$bn_b->swap($bn_a);
```

Exchanges the values of two Crypt::OpenSSL::Bignum objects.

copy

```
my $new_bn_object = $self->copy;
```

Returns a copy of this object.

pointer_copy

```
my $cloned_BIGNUM_ptr = $self->pointer_copy($BIGNUM_ptr);
```

This method is intended only for use by XSUB writers wanting to have access to the underlying BIGNUM structure referenced by a Crypt::OpenSSL::Bignum perl object so that they can pass them to other routines in the OpenSSL library. It returns a perl scalar whose IV can be cast to a BIGNUM* value. This can then be passed to an XSUB which can work with the BIGNUM directly. Note that the

BIGNUM object pointed to will be a copy of the BIGNUM object wrapped by the instance; it is thus the responsibility of the client to free space allocated by this BIGNUM object if and when it is done with it. See also bless_pointer.

AUTHOR

Ian Robertson, iroberts@cpan.org

SEE ALSO

<https://www.openssl.org/docs/crypto/bn.html>