MathCrypto

Release 0.2.4

Adam Ludes

Nov 11, 2022

CONTENTS

1 Includes **3**

2 Installation **5**

3 Getting Started **7**

4 Development **9**

5 License **11**
5.1 Cryptography module 11
5.2 Math module 16

6 Indices and tables **21**

Python Module Index **23**

Index **25**

MathCrypto is a library of useful funtions used in cryptography. Do not use this library for improving the security of your application, it is not safe or powerful enough to provide that.

Current version is 0.2.4.

CHAPTER

ONE

INCLUDES

- **Multiplicative group operations**
  - Generating a group from modulus
  - Get inverse element any element of the group
  - Get element order of any element in group
- **Math functions**
  - Classic number primality check
  - Fermat's primality test
  - Euler's Totient function (Phi)
  - Euclidean algorithm (GCD)
  - Simple number factorization
  - Chinese Remainder Theorem
  - Extended Euclidean Algorithm
- **Cryptography algorithms**
  - **Diffie-Hellman Key exchange generation and cracking**
    - Multithreaded Brute-force cracking
    - Baby-step Giant-step algorithm cracking

CHAPTER

TWO

INSTALLATION

MathCrypto is avalaible through Python Package Index (PyPI) using pip:

```
$ python3 -m pip install mathcrypto
```

To uninstall using pip:

```
$ python3 -m pip uninstall mathcrypto
```

CHAPTER

THREE

GETTING STARTED

```
from mathcrypto import MathFunctions, MultiplicativeGroup

def main():
    print("Playing with math functions")
    print()
    print("Is 137 prime?")
    print(MathFunctions.is_prime(137))
    print("What are the factors of 134?")
    print(MathFunctions.factorize(134))
    print("What is the greatest common divisor of 135 and 186?")
    print(MathFunctions.euclid_gcd(135, 186))
    print("What is the Euler's Totient function of 65?")
    print(MathFunctions.phi(65))
    print()

    print("Playing with multiplicative groups")
    print()
    group = MultiplicativeGroup(13)

    print("Group modulus:")
    print(group.mod)
    print("Group order:")
    print(group.order)
    print("Group elements:")
    print(group.elements)
    print("Group generators:")  # If there are any
    print(group.generators)

main()
```

This outputs:

```
Playing with math functions

Is 137 prime?
True
What are the factors of 134?
[2, 67]
What is the greatest common divisor of 135 and 186?
3
```

(continues on next page)

(continued from previous page)

```
What is the Euler's Totient function of 65?
48

Playing with multiplicative groups

Group modulus:
13
Group order:
12
Group elements:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Group generators:
[2, 6, 7, 11]
```

CHAPTER

FOUR

DEVELOPMENT

Source code repository is available on GitHub. Feel free to contribute. Bug reports and suggestions are welcome.

CHAPTER

FIVE

LICENSE

MathCrypto is licensed under the MIT License.

5.1 Cryptography module

5.1.1 Primes

class mathcrypto.cryptography.primes.**Primes**

Bases: object

classmethod factorize(*num: int*) → list

Classic number factorization

Tests divisibility by 2 and then every odd number up to sqrt(num) while appending the factors. If the number contains multiple instances of a factor, this function returns a list with duplicates. Very fast unless the number is a compound of more than one large prime (more than 7 digits, 8 is still acceptable).

Parameters
num (*int*) – Number to factorize

Returns
List of factors including duplicates

Return type
list

classmethod get_prime(*bit_length: int*) → int

Get a n-bit prime

Parameters
bit_length (*int*) – Bit size of the desired prime number

Returns
desired prime number

Return type
int

classmethod is_prime(*num: int*) → bool

Classic number modulus check

Tests divisibility by 2 and then every odd number up to sqrt(num). Takes longer to compute than Fermat's primality test but has 100% certainty. Cannot handle numbers larger than the system maxint.

Parameters
num (*int*) – Number to test

Raises
OverflowError – If the number is too large

Returns
True if num is prime

Return type
bool

classmethod is_probable_prime_fermat(*num: int*, *rounds: int = 5*) → bool

Automatic Fermat's primality test

This test can not provide 100% certainty that the number is indeed prime, so more than 1 round should be required. If determined that the number is not prime, that is on the other hand 100% certain. Tests for the number of rounds specified, 2-3 rounds are generally enough. If any round returns a result that is not 1, the number is not prime.

Parameters

- **num** (*int*) – Number to be tested
- **rounds** (*int*) – How many rounds of testing to perform
- **verbose** (*bool, optional*) – Whether to return optional list of [<number it was tested against>: *int*,<result>: *int*]. Defaults to False.

Returns
True if probably prime

Return type
bool

If verbose, returns:
(tuple): tuple containing:

- bool: True if probably prime
- list: List of [<number it was tested against>: *int*,<result>: *int*]

5.1.2 Diffie Hellmann

class mathcrypto.cryptography.diffie_hellman.**DHCracker**

Bases: object

classmethod baby_step(*crack_me*) → int

Discrete logarithm problem solution using the Baby-step Giant-step algorithm. RAM intensive.

Parameters
crack_me (*DHCryptosystem object*) – Object containing the publicly know values of the cryptosystem.

Returns
int if a key was found, else None

Return type
int or None

classmethod brute_force(*crack_me*, *num_cpus: int*) → int

Calculates the DHCryptosystem key by utilizing multiprocessing enhanced brute force. CPU and time intensive.

Parameters

- **crack_me** (*DHCryptosystem object*) – Needs to be containing the publicly known values of the cryptosystem.
- **num_cpus** (*int*) – Number of CPU cores to utilize. Do not exceed the number of logical cores your CPU has, this will result in slower execution.

Returns
int if a key was found, else None

Return type
int or None

classmethod mov_attack(*secret: int*, *g: int*, *order: int*) → int

The MOV attack on Elliptic curve DH.

Parameters

- **secret** (*int*) – Secret to be cracked
- **g** (*int*) –
- **order** (*int*) –

Returns
int if secret was cracked, else None

Return type
int or None

class mathcrypto.cryptography.diffie_hellman.**DHCryptosystem**(*prime: Optional[int] = None*, *generator: Optional[int] = None*, *alice_secret: Optional[int] = None*, *bob_secret: Optional[int] = None*, *alice_sends: Optional[int] = None*, *bob_sends: Optional[int] = None*, *key: Optional[int] = None*)

Bases: object

Object containing all values of the cryptosystem

generate_from(*bit_length: Optional[int] = None*, *prime: Optional[int] = None*)

Generates the DHCryptosystem values (If not passed == if they are None) or assigns them

Parameters

- **bit_size** (*int, optional*) – Bit size of the prime. Not necessary if prime also passed.
- **prime** (*int, optional*) – Prime number base of the cryptosystem.

Raises
ValueError – If neither bit_size or prime is passed. At least one of these is required.

generate_rest()

Generates the missing attributes of the DHCryptosystem attributes if possible.

Raises
ValueError – If the attributes that were already in the DHCryptosystem object are calculated from values that were autogenerated. Then you wouldn't have a valid DHCryptosystem.

5.1.3 Elliptic Curves

This module wouldn't be possible to be created this fast if I didn't leverage the code that was written by xnomas and Baka-Git in their repository at:

https://github.com/Baka-Git/Crypto_Math

I want to thank them for allowing me to use their code.

class mathcrypto.cryptography.elliptic_curves.**EllipticCurve**(*a0: int*, *a1: int*, *a2: int*, *a3: int*, *a4: int*, *a5: int*, *a6: int*, *field: Optional[int] = None*, *point_px: Optional[int] = None*, *point_py: Optional[int] = None*)

Bases: `object`

Elliptic curve objects

Parameters

- **a0-a6** (*int*) – Curve attributes (using the equation a0*y^2 + a1*y + a2*y*x = a3*x^3 + a4*x^2 + a5*x+a6)
- **field** (*int, optional*) – The curves field
- **point_px** (*int, optional*) – X coordinate of point P
- **point_py** (*int, optional*) – Y coordinate of point P

add_point(*point_qx: int*, *point_qy: int*)

Adds point P and point Q (of given coordinates) on the curve.

Parameters

- **point_qx** (*int*) – X coordinate of point Q
- **point_qy** (*int*) – Y coordinate of point Q

Raises

- **ValueError** – If curve field is not set.
- **ValueError** – If the curve is not elliptic.
- **ValueError** – If point P is not set.
- **ValueError** – If one or both poits are not on the curve.

Returns
Resulting point coordinates

Return type
list

get_all_point_order()

Gets orders of all points on the curve

Raises
ValueError – If field is not set

Returns
list of lists[order, point]

Return type
list of lists

`get_curve_order`(*get_points: bool = False*)

Gets the order of the curve.

Parameters
get_points (*bool, optional*) – Whether or not to return the curve points as well. Defaults to False.

Raises
ValueError – If curve field is not set or the curve is not elliptic.

Returns

False if this EC is not supported. int: Elliptic curve order if get_points is not set to True (tuple):If get_points is set to true, returns a tuple containing:

- int: Elliptic curve order
- list: List of points on curve

Return type
bool

`get_point_order`(*point_x: Optional[int] = None*, *point_y: Optional[int] = None*)

Gets the order of point of given coordinates or point P if set. Given coordinates take precedence.

Parameters

- **point_x** (*int, optional*) – X coordinate of the point
- **point_y** (*int, optional*) – Y coordinate of the point

Raises
ValueError – Neither valid parameters were passed and point P was not set

Returns
Order of the point

Return type
int

`classmethod get_possible_orders`(*order: int*, *new_order: Optional[int] = None*)

Gets the possible orders of points on a curve of certain order or if a curves order was changed to a given value.

Parameters

- **order** (*int*) – Order of the curve
- **new_order** (*int, optional*) – New order of the curve. Defaults to None.

Returns
List of possible orders

Return type
list

`is_elliptic_curve()`

Checks if the curve is elliptic

Returns
True if curve with these attributes is elliptic

Return type
bool

`is_point_on_elliptic_curve(x: int, y: int)`

Checks if point of given coordinates is on the curve.

Parameters

- **x** (*int*) – X coordinate of the point
- **y** (*int*) – Y coordinate of the point

Raises
ValueError – If curve field is not set or the curve is not elliptic.

Returns
True if point is on the curve

Return type
bool

5.2 Math module

5.2.1 Functions

`class mathcrypto.math.funcs.MathFunctions`

Bases: `object`

A collection of useful mathematical functions

`classmethod crt(lis) → int`

Chinese remainder theorem

Solves x for problems like:
x 8 mod 9

x 3 mod 5

Parameters
lis – list of [int, int]:

Example input:
[[8, 9],[3, 5]] for the example problem

Returns
Solution for x

Return type
int

classmethod eea(*modulus: int*, *number: int*, *verbose: bool = False*) → int

Extended Euclidean Algorithm

Get multiplicative inverse of a number in modulus

Parameters

- **modulus** (*int*) – modulus of a multiplicative group
- **number** (*int*) – number to get inverse to
- **verbose** (*bool, optional*) – Whether to return a graphical solution. Defaults to False.

Raises
ValueError – If number is not an element of the group defined by `modulus`

Returns
resulting inverse

Return type
int

If verbose, returns:
str: Graphical solution of the problem.

classmethod euclid_gcd(*num_a: int*, *num_b: int*) → int

Euclidean algorithm

Calculates the Greatest Common Divisor of two numbers.

Parameters

- **num_a** (*int*) – First number
- **num_b** (*int*) – Second number

Returns
Greatest Common Divisor of the two numbers

Return type
int

classmethod phi(*num: int*) → int

Euler's Totient function Phi. If the number is not prime, the execution time depends on the speed of factorization.

Parameters
num (*int*) – Any positive whole number

Returns
How many elements belong to a multiplicative group set by this number.

Return type
int

5.2.2 Groups

class mathcrypto.math.groups.**MultiplicativeGroup**(*mod*)

Bases: object

Multiplicative group objects

mod

Modulus of the group

Type
int

elements

List of elements in the group

Type
list

order

Order of the group

Type
int

generators

List of generators of the group

Type
list

get_element_order(*element*) → int

Gets the order of an element in the group

Parameters
element (*int*) – Element of the group

Raises
ValueError – When the element does not belong to the group

Returns
Returns the order of element in the group

Return type
int

get_element_subgroup(*element*) → int

Gets the subgroup of any element in the group

Parameters
element (*int*) – Element of the group

Raises
ValueError – When the element does not belong to the group

Returns
Returns the order of element in the group

Return type
list

`get_inverse_element`(*element: int*) → int

Gets the inverse to an element in the group

Parameters
`element` (*int*) – Element of the group

Raises
`ValueError` – When the `element` does not belong to the group

Returns
Inverse element to `element`

Return type
int

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

mathcrypto.cryptography.diffie_hellman, 12
mathcrypto.cryptography.elliptic_curves, 14
mathcrypto.cryptography.primes, 11
mathcrypto.math.funcs, 16
mathcrypto.math.groups, 18

INDEX

A

add_point() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve method*), 14

B

baby_step() (*mathcrypto.cryptography.diffie_hellman.DHCracker class method*), 12
brute_force() (*mathcrypto.cryptography.diffie_hellman.DHCracker class method*), 12

C

crt() (*mathcrypto.math.funcs.MathFunctions class method*), 16

D

DHCracker (*class in mathcrypto.cryptography.diffie_hellman*), 12
DHCryptosystem (*class in mathcrypto.cryptography.diffie_hellman*), 13

E

eea() (*mathcrypto.math.funcs.MathFunctions class method*), 16
elements (*mathcrypto.math.groups.MultiplicativeGroup attribute*), 18
EllipticCurve (*class in mathcrypto.cryptography.elliptic_curves*), 14
euclid_gcd() (*mathcrypto.math.funcs.MathFunctions class method*), 17

F

factorize() (*mathcrypto.cryptography.primes.Primes class method*), 11

G

generate_from() (*mathcrypto.cryptography.diffie_hellman.DHCryptosystem method*), 13
generate_rest() (*mathcrypto.cryptography.diffie_hellman.DHCryptosystem method*), 13
generators (*mathcrypto.math.groups.MultiplicativeGroup attribute*), 18
get_all_point_order() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve method*), 14
get_curve_order() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve method*), 15
get_element_order() (*mathcrypto.math.groups.MultiplicativeGroup method*), 18
get_element_subgroup() (*mathcrypto.math.groups.MultiplicativeGroup method*), 18
get_inverse_element() (*mathcrypto.math.groups.MultiplicativeGroup method*), 18
get_point_order() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve method*), 15
get_possible_orders() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve class method*), 15
get_prime() (*mathcrypto.cryptography.primes.Primes class method*), 11

I

is_elliptic_curve() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve method*), 15
is_point_on_elliptic_curve() (*mathcrypto.cryptography.elliptic_curves.EllipticCurve method*), 16
is_prime() (*mathcrypto.cryptography.primes.Primes class method*), 11
is_probable_prime_fermat() (*mathcrypto.cryptography.primes.Primes class method*), 12

M

mathcrypto.cryptography.diffie_hellman module, 12

mathcrypto.cryptography.elliptic_curves
    module, 14
mathcrypto.cryptography.primes
    module, 11
mathcrypto.math.funcs
    module, 16
mathcrypto.math.groups
    module, 18
MathFunctions (*class in mathcrypto.math.funcs*), 16
mod (*mathcrypto.math.groups.MultiplicativeGroup attribute*), 18
module
    mathcrypto.cryptography.diffie_hellman, 12
    mathcrypto.cryptography.elliptic_curves, 14
    mathcrypto.cryptography.primes, 11
    mathcrypto.math.funcs, 16
    mathcrypto.math.groups, 18
mov_attack() (*mathcrypto.cryptography.diffie_hellman.DHCracker class method*), 13
MultiplicativeGroup (*class in mathcrypto.math.groups*), 18

O

order (*mathcrypto.math.groups.MultiplicativeGroup attribute*), 18

P

phi() (*mathcrypto.math.funcs.MathFunctions class method*), 17
Primes (*class in mathcrypto.cryptography.primes*), 11