crypyto Documentation

Release 0.2.0

Yan Orestes

Aug 22, 2018

API Documentation

crypyto is a Python package that provides a set of cryptographic tools with simple use to your applications.

CHAPTER 1

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

1.1 Dependencies

- Python >= 3.5
- Python packages (no need to worry if you use pip to install crypyto):
  - unidecode to normalize strings
  - Pillow to handle images

1.2 Installing

The easiest way to install crypyto is by using pip:

```
pip install crypyto
```

You can also clone this repository using git

```
git clone https://github.com/yanorestes/crypyto.git
```

CHAPTER 2

Ciphers

This module provides simple usage of functions related to a list of ciphers

Ciphers **crypyto** supports:

- *Polybius Square*
- *Atbash*
- *Caesar Cipher*
- *ROT13*
- *Affine Cipher*
- *Rail Fence Cipher*
- *Keyword Cipher*
- *Vigenère Cipher*
- *Beaufort Cipher*
- *Gronsfeld Cipher*

2.1 Polybius Square

class crypyto.ciphers.**PolybiusSquare**(*width*, *height*, *abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*, *ij=True*)

PolybiusSquare represents a Polybius Square cipher manipulator

Parameters

- **width** (*int*) – The square's width. Must be at least 1. Width times height must be greater than the alphabet length
- **height** (*int*) – The square's height. Must be at least 1. Height times width must be greater than the alphabet length

- **abc** (*str*) – The alphabet used in the square. Defaults to `string.ascii_uppercase`
- **ij** (*bool*) – Whether 'i' and 'j' are treated as the same letter. Defaults to `True`

Raises

- `ValueError` – When *width* is smaller than 1
- `ValueError` – When `width * height` is smaller than `len(abc)`

decrypt(*cipher*)

Returns decrypted cipher (str)

Parameters **cipher** (*str*) – The cipher to be decrypted. May or may not contain the square size at the beggining (e.g. '5x5#')

Raises `ValueError` – When `cipher` doesn't match the Polybius Square pattern

Examples

```
>>> from crypyto.ciphers import PolybiusSquare
>>> ps = PolybiusSquare(5, 5)
>>> ps.decrypt('5x5#5-1;3-3;3-1;2-4;4-5;5-3;4-4;5-1;4-1;2-3;5-1;3-4;3-
↪4;1-1;2-2;5-1')
'ENCRYPTEDMESSAGE'
```

encrypt(*text*)

Returns encrypted text (str)

Parameters **text** (*str*) – The text to be encrypted

Examples

```
>>> from crypyto.ciphers import PolybiusSquare
>>> ps = PolybiusSquare(5, 5)
>>> ps.encrypt('EncryptedMessage')
'5x5#5-1;3-3;3-1;2-4;4-5;5-3;4-4;5-1;4-1;2-3;5-1;3-4;3-4;1-1;2-2;5-1'
```

2.2 Atbash

class crypyto.ciphers.**Atbash**(*abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*)

Atbash represents an Atbash cipher manipulator

Parameters **abc** (*str*) – The alphabet used in the cipher. Defaults to `string.ascii_uppercase`

decrypt(*cipher*, *decode_unicode=True*)

Returns decrypted text (str)

Parameters

- **cipher** (*str*) – The cipher to be decrypted
- **decode_unicode** (*bool*) – Whether the cipher should have unicode characters converted to ascii before decrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Atbash
>>> atbash = Atbash()
>>> atbash.decrypt('SVOOL, DLIOW!')
'HELLO, WORLD!'
```

encrypt(*text*, *decode_unicode=True*)

Returns encrypted text (str)

Parameters

- **text** (*str*) – The text to be encrypted
- **decode_unicode** (*bool*) – Whether the text should have unicode characters converted to ascii before encrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Atbash
>>> atbash = Atbash()
>>> atbash.encrypt('Hello, world!')
'SVOOL, DLIOW!'
```

2.3 Caesar Cipher

class crypyto.ciphers.**Caesar**(*abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*, *key=1*)

Caesar represents a Caesar cipher manipulator

Parameters

- **abc** (*str*) – The alphabet used in the cipher. Defaults to `string.ascii_uppercase`
- **key** (*int*) – The key to initialize the cipher manipulator. Defaults to `1`

brute_force(*cipher*, *decode_unicode=True*, *output_file=None*)

Prints (to stdout or specified file) all possible results

Parameters

- **cipher** (*str*) – The cipher to be decrypted
- **decode_unicode** (*bool*) – Whether the cipher should have unicode characters converted to ascii before decrypting. Defaults to `True`
- **output_file** (*str|None*) – The filename of the file the results are gonna be printed. Defaults to `None`, which indicated printing on stdout

Examples

```
>>> from crypyto.ciphers import Caesar
>>> caesar = Caesar()
>>> caesar.brute_force('MJQQT, BTWQI!')
NKRRU, CUXRJ!
OLSSV, DVYSK!
...
```

(continues on next page)

(continued from previous page)

```
HELLO, WORLD!
IFMMP, XPSME!
...
```

decrypt(*cipher*, *decode_unicode=True*, *key=None*)

Returns decrypted cipher (str)

Parameters

- **cipher** (*str*) – The cipher to be decrypted
- **decode_unicode** (*bool*) – Whether the cipher should have unicode characters converted to ascii before decrypting. Defaults to `True`
- **key** (*int|None*) – The key used to decrypt. Defaults to `None`, which uses the value from `self.key`

Examples

```
>>> from crypyto.ciphers import Caesar
>>> caesar = Caesar(key=5)
>>> caesar.decrypt('MJQQT, BTWQI!')
'HELLO, WORLD!'
```

encrypt(*text*, *decode_unicode=True*, *key=None*)

Returns encrypted text (str)

Parameters

- **text** (*str*) – The text to be encrypted
- **decode_unicode** (*bool*) – Whether the text should have unicode characters converted to ascii before encrypting. Defaults to `True`
- **key** (*int|None*) – The key used to encrypt. Defaults to `None`, which uses the value from `self.key`

Examples

```
>>> from crypyto.ciphers import Caesar
>>> caesar = Caesar(key=5)
>>> caesar.encrypt('Hello, world!')
'MJQQT, BTWQI!'
```

2.4 ROT13

A `Caesar` object with default `key` value of `13`

Examples:

```
>>> from crypyto.ciphers import ROT13
>>> ROT13.encrypt('Hello, world!')
'URYYB, JBEYQ!'
>>> ROT13.encrypt('URYYB, JBEYQ!')
'HELLO, WORLD!'
```

2.5 Affine Cipher

class crypyto.ciphers.**Affine**(*a*, *b*, *abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*)

Affine represents an Affine cipher manipulator

Parameters

- **a** (*int*) – Value of `a`. Must be coprime to `len(abc)`
- **b** (*int*) – Value of `b`
- **abc** (*str*) – The alphabet used in the cipher. Defaults to `string.ascii_uppercase`

Raises `ValueError` – If `a` is not coprime to `len(abc)`

decrypt(*cipher*)

Returns decrypted cipher (str)

Parameters **cipher** (*str*) – Cipher to be decrypted

Examples

```
>>> from crypyto.cipher import Affine
>>> af = Affine(a=5, b=8)
>>> af.decrypt('RCLLA, OAPLX!')
'HELLO, WORLD!'
```

encrypt(*text*)

Returns encrypted text (str)

Parameters **text** (*str*) – Text to be encrypted

Examples

```
>>> from crypyto.cipher import Affine
>>> af = Affine(a=5, b=8)
>>> af.encrypt('Hello, world!')
'RCLLA, OAPLX!'
```

2.6 Rail Fence Cipher

class crypyto.ciphers.**RailFence**(*n_rails*, *only_alnum=False*, *direction='D'*)

RailFence represents a Rail Fence cipher manipulator

Parameters

- **n_rails** (*int*) – Number of rails
- **only_alnum** (*bool*) – Whether the manipulator will only encrypt alphanumerical characters. Defaults to `False`
- **direction** (*str*) – Default direction to start zigzagging. Must be `'D'` (Downwards) or `'U'` (Upwards). Defaults to `'D'`

Raises `ValueError` – When `direction` doesn't start with `'U'` or `'D'`

brute_force(*cipher*, *output_file=None*)
Prints (to stdout or specified file) all possible decrypted results

Parameters
- **cipher** (*str*) – The cipher to be decrypted
- **output_file** (*str|None*) – The filename of the file the results are gonna be printed. Defaults to None, which indicated printing on stdout

Examples

```
>>> from crypyto.ciphers import RailFence
>>> rf = RailFence(n_rails=1, only_alnum=True)
>>> rf.decrypt('WECRLTEERDSOEEFEAOCAIVDEN')
There are 46 possible results. You can specify an output file in the␣
↪parameter output_file
Are you sure you want to print them all (Y/N)?
Y
WEEFCERALOTCEAEIRVDDSEONE
WEAREDISCOVEREDFLEEATONCE
...
NEDVIACOAEFEEOSDREETREWCL
NEDVIACOAEFEEOSDREETLREWC
```

decrypt(*cipher*)
Returns decrypted cipher

Parameters **cipher** (*str*) – The cipher to be decrypted

Examples

```
>>> from crypyto.cipher import RailFence
>>> rf = RailFence(n_rails=3, only_alnum=True)
>>> rf.decrypt('WECRLTEERDSOEEFEAOCAIVDEN')
'WEAREDISCOVEREDFLEEATONCE'
```

encrypt(*text*)
Returns encrypted text (str)

Parameters **text** (*str*) – The text to be encrypted

Examples

```
>>> from crypyto.cipher import RailFence
>>> rf = RailFence(n_rails=3, only_alnum=True)
>>> rf.encrypt('WE ARE DISCOVERED. FLEE AT ONCE')
'WECRLTEERDSOEEFEAOCAIVDEN'
```

2.7 Keyword Cipher

class crypyto.ciphers.**Keyword**(*key*, *abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*)
Keyword represents a Keyword Cipher manipulator

Parameters

- **key** (*str*) – The keyword to encrypt/decrypt
- **abc** (*str*) – The alphabet used in the cipher. Defaults to `string.ascii_uppercase`

decrypt(*cipher*)
Returns decrypted cipher (str)
Parameters **cipher** (*str*) – Cipher to be decrypted

Examples

```
>>> from crypyto.ciphers import Keyword
>>> kw = Keyword('secret')
>>> kw.decrypt('BEHHK, VKNHR!')
'HELLO, WORLD!'
```

encrypt(*text*)
Returns encrypted text (str)
Parameters **text** (*str*) – Text to be encrypted

Examples

```
>>> from crypyto.ciphers import Keyword
>>> kw = Keyword('secret')
>>> kw.encrypt('Hello, world!')
'BEHHK, VKNHR!'
```

2.8 Vigenère Cipher

class crypyto.ciphers.**Vigenere**(*key*, *abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*, *decode_unicode_key=True*)
Vigenere represents a Vigenère Cipher manipulator

Parameters

- **key** (*str*) – The key to encode/decode
- **abc** (*str*) – The alphabet the cipher will be based upon. Defauts to `string.ascii.uppercase`
- **decode_unicode_key** (*bool*) – Whether the key should have unicode characters converted to ascii. Defaults to `True`

decrypt(*cipher*, *decode_unicode=True*)
Returns decrypted cipher
Parameters
- **cipher** (*str*) – Cipher to be decrypted
- **decode_unicode** (*bool*) – Whether the cipher should have unicode characters converted to ascii before decrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Vigenere
>>> v = Vigenere('secret')
>>> v.decrypt('ZINCS, PGVNU!')
'HELLO, WORLD!'
```

encrypt(*text*, *decode_unicode=True*)

Returns encrypted text (str)

Parameters

- **text** (*str*) – Text to be encrypted
- **decode_unicode** (*bool*) – Whether the text should have unicode characters converted to ascii before encrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Vigenere
>>> v = Vigenere('secret')
>>> v.encrypt('Hello, world!')
'ZINCS, PGVNU!'
```

2.9 Beaufort Cipher

class crypyto.ciphers.**Beaufort**(*key*, *abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*, *decode_unicode_key=True*)

Beaufort represents a Beaufort Cipher manipulator

Parameters

- **key** (*str*) – The key to encode/decode
- **abc** (*str*) – The alphabet the cipher will be based upon. Defauts to `string.ascii.uppercase`
- **decode_unicode_key** (*bool*) – Whether the key should have unicode characters converted to ascii. Defaults to `True`

decrypt(*cipher*, *decode_unicode=True*)

Returns decrypted cipher (str)

Parameters

- **cipher** (*str*) – The cipher to be decrypted
- **decode_unicode** (*bool*) – Whether the cipher should have unicode characters converted to ascii before decrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Beaufort
>>> b = Beaufort('secret')
>>> b.decrypt('LARGQ, XENRO!')
'HELLO, WORLD!'
```

encrypt(*text*, *decode_unicode=True*)
Returns encrypted text (str)

Parameters

- **text** (*str*) – The text to be encrypted
- **decode_unicode** (*bool*) – Whether the text should have unicode characters converted to ascii before encrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Beaufort
>>> b = Beaufort('secret')
>>> b.encrypt('Hello, world!')
'LARGQ, XENRO!'
```

2.10 Gronsfeld Cipher

class crypyto.ciphers.**Gronsfeld**(*key*, *abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'*)
Gronsfeld represents a Gronsfeld Cipher manipulator

Parameters

- **key** (*str*) – The key to encode/decode. Must contain only numerical characters (0-9)
- **abc** (*str*) – The alphabet the cipher will be based upon. Defauts to `string.ascii.uppercase`

decrypt(*cipher*, *decode_unicode=True*)
Returns decrypted cipher (str)

Parameters

- **cipher** (*str*) – The cipher to be decrypted
- **decode_unicode** (*bool*) – Whether the text should have unicode characters converted to ascii before encrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Gronsfeld
>>> g = Gronsfeld('2317')
>>> g.decrypt('JHMSQ, ZPYNG!')
'HELLO, WORLD!'
```

encrypt(*text*, *decode_unicode=True*)
Returns encrypted text (str)

Parameters

- **text** (*str*) – The text to be encrypted
- **decode_unicode** (*bool*) – Whether the text should have unicode characters converted to ascii before encrypting. Defaults to `True`

Examples

```
>>> from crypyto.ciphers import Gronsfeld
>>> g = Gronsfeld('2317')
>>> g.encrypt('Hello, world!')
'JHMSQ, ZPYNG!'
```

CHAPTER 3

Substitution Alphabets

This module provides simple usage of functions related to substitutions alphabets

Alphabets **crypyto** supports:

- *Morse Code*
- *Binary Translation*
- *Pigpen Cipher*
- *Templar Cipher*
- *Betamaze Alphabet*

3.1 Morse Code

class crypyto.substitution_alphabets.**Morse**(*word_splitter='/'*)

Morse represents a Morse Code manipulator

Parameters **word_splitter** (*str*) – A string which will be used to indicate words separation. Defaults to `'/'`

decrypt(*cipher*)

Returns translated cipher into plain text

Parameters **cipher** (*str*) – The morse code to be translated into plain text

Examples

```
>>> from crypyto.substitution_alphabets import Morse
>>> morse = Morse()
>>> morse.decrypt('.... . .-.. .-.. --- --..-- / .-- --- .-. .-.. -..␣
↪-.-.--')
'HELLO, WORLD!'
```

encrypt(*text*)
Returns translated text into Morse Code (str)
Parameters **text** (*str*) – The text to be translated into Morse Code

Examples

```
>>> from crypyto.subsititution_alphabets import Morse
>>> morse = Morse()
>>> morse.encrypt('Hello, world!')
'.... . .-.. .-.. --- --..-- / .-- --- .-. .-.. -.. -.-.--'
```

3.2 Binary Translation

class crypyto.substitution_alphabets.**Binary**(*letter_splitter=' '*)
Binary represents a text-to-binary manipulator

Parameters **letter_splitter** (*str*) – A string which will be used to indicate characters separation. Defaults to ' '

decrypt(*cipher*)
Returns the binary-cipher translated to text
Parameters **cipher** (*str*) – The binary-cipher to be translated to normal text

Examples

```
>>> from crypyto.substitution_alphabets import Binary
>>> b = Binary()
>>> b.decrypt('1001000 1100101 1101100 1101100 1101111 101100 100000␣
↪1110111 1101111 1110010 1101100 1100100 100001')
'Hello, world!'
```

encrypt(*text*)
Returns the text translated to binary
Parameters **text** (*str*) – The text to be translated to binary

Examples

```
>>> from crypyto.substitution_alphabets import Binary
>>> b = Binary()
>>> b.encrypt('Hello, world!')
'1001000 1100101 1101100 1101100 1101111 101100 100000 1110111␣
↪1101111 1110010 1101100 1100100 100001'
```

3.3 Pigpen Cipher

class crypyto.substitution_alphabets.**Pigpen**
Pigpen represents a Pigpen Cipher manipulator

decrypt(*filename*)

Returns the image cipher translated to normal text (str). It will most often do it wrong, because of specifications on Pigpen. I'll try to fix that soon.

Parameters **filename** (*str*) – Filename of the cipher image file

Raise: ValueError: If the size of the respective image doensn't match the cipher pattern. I'll try to work on that.

Examples

```
>>> from crypyto.substitution_alphabets import Pigpen
>>> pigpgen = Pigpen()
>>> pigpen.decrypt('pigpen_hello.png')
'BEJJMWMJJD'
>>> pigpen.decrypt('pigpen_hello_max.png')
'BEJJMWMJJD'
```

encrypt(*text*, *filename='output.png'*, *max_in_line=30*)

Creates an image file with the translated text

Parameters

- **text** (*str*) – Text to be translated to the Pigpen alphabet
- **filename** (*str*) – The filename of the image file with the translated text. Defaults to `'output.png'`
- **max_in_line** (*int*) – The max number of letters per line. Defaults to `30`

Examples

```
>>> from crypyto.substitution_alphabets import Pigpen
>>> pigpen = Pigpen()
>>> pigpen.encrypt('Hello, world!', 'pigpen_hello.png')
>>> pigpen.encrypt('Hello, world!', 'pigpen_hello_max.png', 5)
```

pigpen_hello.png:

Fig. 1: Encrypted hello world

pigpen_hello_max.png:

Fig. 2: Encrypted hello world (5 letters per line)

3.4 Templar Cipher

class crypyto.substitution_alphabets.**Templar**

Templar represents a Templar Cipher manipulator

decrypt(*filename*)

Returns the image cipher translated to normal text (str)

Parameters **filename** (*str*) – Filename of the cipher image file

Raise: ValueError: If the size of the respective image doensn't match the cipher pattern. I'll try to work on that.

Examples

```
>>> from crypyto.substitution_alphabets import Templar
>>> templar = Templar()
>>> templar.decrypt('templar_hello.png')
'HELLOWORLD'
>>> templar.decrypt('templar_hello_max.png')
'HELLOWORLD'
```

encrypt(*text*, *filename='output.png'*, *max_in_line=30*)

Creates an image file with the translated text

Parameters

- **text** (*str*) – Text to be translated to the Templar alphabet
- **filename** (*str*) – The filename of the image file with the translated text. Defaults to `'output.png'`
- **max_in_line** (*int*) – The max number of letters per line. Defaults to `30`

Examples

```
>>> from crypyto.substitution_alphabets import Templar
>>> templar = Templar()
>>> templar.encrypt('Hello, world!', 'templar_hello.png')
>>> templar.encrypt('Hello, world!', 'templar_hello_max.png', 5)
```

templar_hello.png:

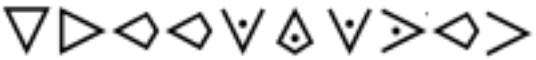

Fig. 3: Encrypted hello world

templar_hello_max.png:

Fig. 4: Encrypted hello world (5 letters per line)

3.5 Betamaze Alphabet

class crypyto.substitution_alphabets.**Betamaze**(*random_rotate=False*)

Betamaze represents a Betamaze Alphabet Manipulator

Parameters **random_rotate** (*bool*) – Whether to randomly rotate each square letter (as it is possible with Betamaze). Defaults to `False`

decrypt(*filename*)

Returns the image cipher translated to normal text (str)

Parameters **filename** (*str*) – Filename of the cipher image file

Raise: ValueError: If the size of the respective image doensn't match the cipher pattern. I'll try to work on that.

Examples

```
>>> from crypyto.substitution_alphabets import Betamaze
>>> betamaze = Betamaze()
>>> betamaze.decrypt('betamaze_hello.png')
'HELLO, WORLD'
>>> betamaze.decrypt('betamaze_hello_random.png')
'HELLO, WORLD'
```

encrypt(*text*, *filename='output.png'*, *max_in_line=10*)

Creates an image file with the translated text

Parameters

- **text** (*str*) – Text to be translated to the Betamaze alphabet
- **filename** (*str*) – The filename of the image file with the translated text. Defaults to `'output.png'`
- **max_in_line** (*int*) – The max number of letters per line. Defaults to `10`

Examples

```
>>> from crypyto.substitution_alphabets import Betamaze
>>> betamaze = Betamaze()
>>> betamaze.encrypt('Hello, world!', 'betamaze_hello.png', 5)
>>>
>>> betamaze_random = Betamaze(random_rotate=True)
>>> betamaze_random.encrypt('Hello, world!', 'betamaze_hello_random.
↪png', 5)
```

betamaze_hello.png:

betamaze_hello_random.png:

Fig. 5: Encrypted hello world (5 letters per line)

Fig. 6: Encrypted hello world (5 letters per line and with random rotation)

Python Module Index

C

Index