pycipher Documentation

Release 1

James Lyons

Apr 30, 2017

Contents

Common classical ciphers implemented in Python.

The code is hosted on GitHub: https://github.com/jameslyons/pycipher

If you find any bugs make an issue or create a pull request.

CHAPTER 1

Example usage

```
>>> from pycipher import ADFGVX
>>> adfgvx = ADFGVX(key='PH0QG64MEA1YL2NOFDXKR3CVS5ZW7BJ9UTI8', keyword='GERMAN')
>>> adfgvx.encipher("Hello world!")
'FVFDAGXAFFFFGFAGADFG'
>>> adfgvx.decipher(_)
'HELLOWORLD'
```

Table of Contents

CHAPTER 2

A Short Aside on Keysquares

Many of the ciphers mentioned here e.g. *pycipher.Playfair()*, *pycipher.Foursquare()* use keysquares as part of their key information. A keysquare looks like this:

```
Z G P T F
O I H M U
W D R C N
Y K E Q A
X V S B L
```

The keysquare above is a 5 by 5 keysquare consisting of 25 characters. To use this keysquare as part of a key, read each row starting with the top row, then the second row etc. The keysquare above becomes "ZGPTFOIHMUWDRCNYKEQAXVSBL". Note that with only 25 characters, 1 character can't be included. This is usually the letter 'J', wherever this letter appears it is replaced by 'I'.

Keysquares can also be 6 by 6. In this case the 26 letters A-Z plus the 10 numbers 0-9 are used. An example of this sort of keysquare is:

```
p h 0 q g 6
4 m e a 1 y
l 2 n o f d
x k r 3 c v
s 5 z w 7 b
j 9 u t i 8
```

The keysquare above becomes "ph0qg64mea1yl2nofdxkr3cvs5zw7bj9uti8".

CHAPTER 3

The Ciphers

ADFGX Cipher

This cipher uses a keysquare as part of its key, see *A Short Aside on Keysquares* for information.

class pycipher.**ADFGX**(*key='phqgmeaylnofdxkrcvszwbuti'*, *keyword='GERMAN'*)

The ADFGX Cipher has a key consisting of a 5x5 key square and a word e.g. 'GERMAN'. The algorithm is described here: http://www.practicalcryptography.com/ciphers/classical-era/adfgvx/ The key square consists of the letters A-Z with J omitted (25 characters total).

Parameters

- **key** – The keysquare, as a 25 character string.
- **keyword** – The keyword, any word or phrase will do.

decipher(*string*)

Decipher string using ADFGX cipher according to initialised key information. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = ADFGX('phqgmeaylnofdxkrcvszwbuti','HELLO').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The enciphered string.

encipher(*string*)

Encipher string using ADFGX cipher according to initialised key information. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = ADFGX('phqgmeaylnofdxkrcvszwbuti','HELLO').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

ADFGVX Cipher

This cipher uses a keysquare as part of its key, see *A Short Aside on Keysquares* for information.

class pycipher.**ADFGVX**(*key='ph0qg64mea1yl2nofdxkr3cvs5zw7bj9uti8'*, *keyword='GERMAN'*)

The ADFGVX Cipher has a key consisting of a 6x6 key square and a word e.g. 'GERMAN'. The algorithm is described here: http://www.practicalcryptography.com/ciphers/classical-era/adfgvx/ The key square consists of the letters A-Z and the numbers 0-9 (36 characters total).

Parameters

- **key** – The keysquare, as a 36 character string.
- **keyword** – The keyword, any word or phrase will do.

decipher(*string*)

Decipher string using ADFGVX cipher according to initialised key information. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = ADFGVX('ph0qg64mea1yl2nofdxkr3cvs5zw7bj9uti8','HELLO').
↪decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The enciphered string.

encipher(*string*)

Encipher string using ADFGVX cipher according to initialised key information. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = ADFGVX('ph0qg64mea1yl2nofdxkr3cvs5zw7bj9uti8','HELLO').
↪encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Affine Cipher

class pycipher.**Affine**(*a=5*, *b=9*)

The Affine Cipher has two components to the key, numbers a and b. This cipher encrypts a letter according to the following equation:

```
c = (a*p + b)%26
```

where c is the ciphertext letter, p the plaintext letter. b is an integer 0-25, a is an integer that has an inverse (mod 26). Allowable values for a are: 1,3,5,7,9,11,15,17,19,21,23,25 For more info on the Affine cipher see http://www.practicalcryptography.com/ciphers/affine-cipher/

Parameters

- **a** – The multiplicative part of the key. Allowable values are: 1,3,5,7,9,11,15,17,19,21,23,25
- **b** – The additive part of the key. Allowable values are integers 0-25

decipher(*string*, *keep_punct=False*)

Decipher string using affine cipher according to initialised key.

Example:

```
plaintext = Affine(a,b).decipher(ciphertext)
```

Parameters

- **string** – The string to decipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The deciphered string.

encipher(*string*, *keep_punct=False*)

Encipher string using affine cipher according to initialised key.

Example:

```
ciphertext = Affine(a,b).encipher(plaintext)
```

Parameters

- **string** – The string to encipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The enciphered string.

Autokey Cipher

class pycipher.**Autokey**(*key='FORTIFICATION'*)

The Autokey Cipher has a key consisting of a word e.g. 'FORTIFICATION'. This cipher encrypts a letter according to the Vigenere tableau, the algorithm can be seen e.g. http://www.practicalcryptography.com/ciphers/classical-era/autokey/

Parameters **key** – The keyword, any word or phrase will do. Must consist of alphabetical characters only, no punctuation of numbers.

decipher(*string*)

Decipher string using Autokey cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = Autokey('HELLO').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The enciphered string.

encipher(*string*)
Encipher string using Autokey cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Autokey('HELLO').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Atbash Cipher

class pycipher.**Atbash**
The Atbash Cipher has no key. For more information see http://www.practicalcryptography.com/ciphers/atbash-cipher-cipher/.

decipher(*string*, *keep_punct=False*)
Decipher string using the Atbash cipher.

Example:

```
plaintext = Atbash().decipher(ciphertext)
```

Parameters

- **string** – The string to decipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The deciphered string.

encipher(*string*, *keep_punct=False*)
Encipher string using Atbash cipher.

Example:

```
ciphertext = Atbash().encipher(plaintext)
```

Parameters

- **string** – The string to encipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The enciphered string.

Beaufort Cipher

class pycipher.**Beaufort**(*key='FORTIFICATION'*)
The Beaufort Cipher is similar to the Vigenere Cipher, and has a key consisting of a word e.g. 'FORTIFICATION'. This cipher encrypts a letter according to the Vigenere tableau, the but uses a different algorithm

to find the ciphertext letter. The algorithm can be seen e.g. http://www.practicalcryptography.com/ciphers/beaufort-cipher/

Parameters **key** – The keyword, any word or phrase will do. Must consist of alphabetical characters only, no punctuation of numbers.

decipher(*string*)

Decipher string using Beaufort cipher according to initialised key. Punctuation and whitespace are removed from the input. For the Beaufort cipher, enciphering and deciphering are the same operation.

Example:

```
plaintext = Beaufort('HELLO').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)

Encipher string using Beaufort cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Beaufort('HELLO').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Bifid Cipher

This cipher uses a keysquare as part of its key, see *A Short Aside on Keysquares* for information.

class pycipher.**Bifid**(*key='phqgmeaylnofdxkrcvszwbuti'*, *period=5*)

The Bifid Cipher is a fractionating cipher, and has a key consisting of a 25 letter keysquare (with a letter removed e.g. 'J'), along with a 'period', which is an integer. For more information, the algorithm can be seen e.g. http://www.practicalcryptography.com/ciphers/bifid-cipher/

Parameters

- **key** – The keysquare, as a 25 character string.
- **period** – an integer.

decipher(*string*)

Decipher string using Bifid cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = Bifid('phqgmeaylnofdxkrcvszwbuti',5).decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)
Encipher string using Bifid cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Bifid('phqgmeaylnofdxkrcvszwbuti',5).encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Caesar Cipher

class pycipher.**Caesar**(*key=13*)
The Caesar Cipher has a key consisting of an integer 1-25. This cipher encrypts a letter according to the following equation:

```
c = (p + key)%26
```

where c is the ciphertext letter, p the plaintext letter. For more details on the Caesar cipher, see http://www.practicalcryptography.com/ciphers/caesar-cipher/

Parameters **key** – The additive key. Allowable values are integers 0-25.

decipher(*string*, *keep_punct=False*)
Decipher string using Caesar cipher according to initialised key.

Example:

```
plaintext = Caesar(3).decipher(ciphertext)
```

Parameters

- **string** – The string to decipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The deciphered string.

encipher(*string*, *keep_punct=False*)
Encipher string using Caesar cipher according to initialised key.

Example:

```
ciphertext = Caesar(3).encipher(plaintext)
```

Parameters

- **string** – The string to encipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The enciphered string.

Columnar Transposition Cipher

class pycipher.**ColTrans**(*keyword='GERMAN'*)

The Columnar Transposition Cipher is a fractionating cipher, and has a key consisting of a word e.g. 'GERMAN' For more information, the algorithm can be seen e.g. http://www.practicalcryptography.com/ciphers/columnar-transposition-cipher/ .

Parameters **key** – The keyword, any word or phrase will do. Must consist of alphabetical characters only, no punctuation of numbers.

decipher(*string*)

Decipher string using Columnar Transposition cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = ColTrans('GERMAN').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)

Encipher string using Columnar Transposition cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = ColTrans('GERMAN').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Enigma M3 Cipher

The Enigma M3 was used by the German army during the Second World War. The keying information for mechanical ciphers can be a little more complicated than for the simpler ciphers, it may be beneficial to read this guide to get an understanding of how the Enigma works before using it.

class pycipher.**Enigma**(*settings=('A', 'A', 'A'), rotors=(1, 2, 3), reflector='B', ringstellung=('F', 'V', 'N'), steckers=[('P', 'O'), ('M', 'L'), ('I', 'U'), ('K', 'J'), ('N', 'H'), ('Y', 'T'), ('G', 'B'), ('V', 'F'), ('R', 'E'), ('D', 'C')]*)

The Enigma M3 cipher. The key consists of several parameters.

Parameters

- **settings** – The rotor start positions, consists of 3 characters e.g. ('V','B','Q')
- **rotors** – The rotors and their order e.g. (2,3,1). There are 8 possible rotors.
- **reflector** – The reflector in use, a single character 'A','B' or 'C'
- **ringstellung** – The ring settings, consists of 3 characters e.g. ('V','B','Q')
- **steckers** – The plugboard settings, indicating which characters are replaced by which others. Consists of a 10-tuple of 2-tuples e.g. [('P','O'), ('M','L'), ('I','U'), ('K','J'),

('N','H'), ('Y','T'), ('G','B'), ('V','F'), ('R','E'), ('D','C')]. If fewer plugs are required leave them out e.g. a 5-tuple of 2-tuples would be used if 5 plugs were applied.

decipher(*string*)

Decipher string using Enigma M3 cipher according to initialised key. Punctuation and whitespace are removed from the input. The encipher and decipher operations of the Enigma are identical.

Example:

```
plaintext = Enigma(settings=('A','A','A'),rotors=(1,2,3),reflector='B',
    ringstellung=('F','V','N'),steckers=[('P','O'),('M','L'),
    ('I','U'),('K','J'),('N','H'),('Y','T'),('G','B'),('V','F'),
    ('R','E'),('D','C')])).decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)

Encipher string using Enigma M3 cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Enigma(settings=('A','A','A'),rotors=(1,2,3),reflector='B',
    ringstellung=('F','V','N'),steckers=[('P','O'),('M','L'),
    ('I','U'),('K','J'),('N','H'),('Y','T'),('G','B'),('V','F'),
    ('R','E'),('D','C')])).encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Foursquare Cipher

This cipher uses a keysquare as part of its key, see *A Short Aside on Keysquares* for information.

class pycipher.**Foursquare**(*key1='zgptfoihmuwdrcnykeqaxvsbl'*, *key2='mfnbdcrhsaxyogvituewlqzkp'*)

The Foursquare Cipher enciphers pairs of characters, the key consists of 2 keysquares, each 25 characters in length. More information about the algorithm can be found at http://www.practicalcryptography.com/ciphers/four-square-cipher/.

Parameters

- **key1** – The first keysquare, as a 25 character string.
- **key2** – The second keysquare, as a 25 character string.

decipher(*string*)

Decipher string using Foursquare cipher according to initialised key. Punctuation and whitespace are removed from the input. The ciphertext should be an even number of characters. If the input ciphertext is not an even number of characters, an 'X' will be appended.

Example:

```
plaintext = Foursquare(key1='zgptfoihmuwdrcnykeqaxvsbl',key2=
↪'mfnbdcrhsaxyogvituewlqzkp').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)

Encipher string using Foursquare cipher according to initialised key. Punctuation and whitespace are removed from the input. If the input plaintext is not an even number of characters, an 'X' will be appended.

Example:

```
ciphertext = Foursquare(key1='zgptfoihmuwdrcnykeqaxvsbl',key2=
↪'mfnbdcrhsaxyogvituewlqzkp').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Gronsfeld Cipher

class pycipher.**Gronsfeld**(*key=[5, 4, 7, 9, 8, 5, 8, 2, 0, 9, 8, 4, 3]*)

The Gronsfeld Cipher is similar to the Vigenere Cipher, and has a key consisting of a sequence of numbers 0-9 e.g. [4,9,2,0,2]. This cipher encrypts a letter according to the Vigenere tableau. More information about the algorithm can be found at http://www.practicalcryptography.com/ciphers/vigenere-gronsfeld-and-autokey-cipher/

Parameters **key** – The keyword, any word or phrase will do. Must consist of alphabetical characters only, no punctuation of numbers.

decipher(*string*)

Decipher string using Gronsfeld cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = Gronsfeld([5, 4, 7, 9, 8, 5, 8, 2, 0, 9, 8, 4, 3]).
↪decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)

Encipher string using Gronsfeld cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Gronsfeld([5, 4, 7, 9, 8, 5, 8, 2, 0, 9, 8, 4, 3]).
↪encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

M-209 Cipher

The M-209 was used by the American army during the Second World War. The keying information for mechanical ciphers can be a little more complicated than for the simpler ciphers, it may be beneficial to read this page , in particular the parts about "Example Configuration" to get an understanding of how the M-209 works before using it.

class pycipher.**M209**(*wheel_starts='AAAAAA'*, *w1s=None*, *w2s=None*, *w3s=None*, *w4s=None*, *w5s=None*, *w6s=None*, *lugpos=None*)

The M209 cipher. The key consists of several parameters.

Parameters

- **wheel_starts** – The rotor start positions, consists of 6 characters e.g. "AAAAAA". Note that not all character combinations are possible, e.g. wheel 6 has only 17 characters.
- **w1s** – wheel 1 settings. An array of 26 binary values.
- **w2s** – wheel 2 settings. An array of 25 binary values.
- **w3s** – wheel 3 settings. An array of 23 binary values.
- **w4s** – wheel 4 settings. An array of 21 binary values.
- **w5s** – wheel 5 settings. An array of 19 binary values.
- **w6s** – wheel 6 settings. An array of 17 binary values.
- **lugpos** – The lugs, a 27-tuple of 2-tuples. See below for an example.

Example:

```
wheel_1_settings = [1,1,0,1,0,0,0,1,1,0,1,0,1,1,0,0,0,0,1,1,0,1,1,0,0,0]
wheel_2_settings = [1,0,0,1,1,0,1,0,0,1,1,1,0,0,1,0,0,1,1,0,1,0,1,0,0]
wheel_3_settings = [1,1,0,0,0,0,1,1,0,1,0,1,1,1,0,0,0,1,1,1,1,0,1]
wheel_4_settings = [0,0,1,0,1,1,0,1,1,0,0,0,1,1,0,1,0,0,1,1,1]
wheel_5_settings = [0,1,0,1,1,1,0,1,1,0,0,0,1,1,0,1,0,0,1]
wheel_6_settings = [1,1,0,1,0,0,0,1,0,0,1,0,0,1,1,0,1]
wheel_starts = "AAAAAA"
lug_positions = ((0,6),(3,6),(1,6),(1,5),(4,5),(0,4),(0,4),(0,4),(0,4),
                 (2,0),(2,0),(2,0),(2,0),(2,0),(2,0),(2,0),(2,0),(2,0),
                 (2,0),(2,5),(2,5),(0,5),(0,5),(0,5),(0,5),(0,5),(0,5))
m = M209(wheel_1_settings, wheel_2_settings, wheel_3_settings
         wheel_4_settings, wheel_5_settings, wheel_6_settings
         wheel_starts, lug_positions)
```

decipher(*message*)

Decipher string using M209 cipher according to initialised key. Punctuation and whitespace are removed from the input. The encipher and decipher operations of the M209 are identical.

Example (continuing from the example above):

```
plaintext = m.decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*message*)

Encipher string using M209 cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example (continuing from the example above):

```
ciphertext = m.encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Playfair Cipher

This cipher uses a keysquare as part of its key, see *A Short Aside on Keysquares* for information.

class pycipher.**Playfair**(*key='ABCDEFGHIKLMNOPQRSTUVWXYZ'*)

The Playfair Cipher enciphers pairs of characters, the key consists of a keysquare 25 characters in length. More information about the algorithm can be found at http://www.practicalcryptography.com/ciphers/playfair-cipher/.

Parameters **key** – The keysquare, as a 25 character string.

decipher(*string*)

Decipher string using Playfair cipher according to initialised key. Punctuation and whitespace are removed from the input. The ciphertext should be an even number of characters. If the input ciphertext is not an even number of characters, an 'X' will be appended.

Example:

```
plaintext = Playfair(key='zgptfoihmuwdrcnykeqaxvsbl').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)

Encipher string using Playfair cipher according to initialised key. Punctuation and whitespace are removed from the input. If the input plaintext is not an even number of characters, an 'X' will be appended.

Example:

```
ciphertext = Playfair(key='zgptfoihmuwdrcnykeqaxvsbl').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Polybius Square Cipher

This cipher uses a keysquare as part of its key, see *A Short Aside on Keysquares* for information.

class pycipher.**PolybiusSquare**(*key='phqgiumeaylnofdxkrcvstzwb'*, *size=5*, *chars=None*)

The Polybius square is a simple substitution cipher that outputs 2 characters of ciphertext for each character of plaintext. It has a key consisting which depends on 'size'. By default 'size' is 5, and the key is 25 letters (5^2). For a size of 6 a 36 letter key required etc. For a more detailed look at how it works see http://www.practicalcryptography.com/ciphers pycipher.polybius-square-cipher/.

Parameters

- **key** – The keysquare, each row one after the other. The key must by size^2 characters in length.

- **size** – The size of the keysquare, if size=5, the keysquare uses 5^2 or 25 characters.
- **chars** – the set of characters to use. By default ABCDE are used, this parameter should have the same length as size.

decipher(*string*)
Decipher string using Polybius square cipher according to initialised key.

Example:

```
plaintext = Polybius('APCZWRLFBDKOTYUQGENHXMIVS',5,'MKSBU').
↪decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string. The plaintext will be half the length of the ciphertext.

encipher(*string*)
Encipher string using Polybius square cipher according to initialised key.

Example:

```
ciphertext = Polybius('APCZWRLFBDKOTYUQGENHXMIVS',5,'MKSBU').
↪encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string. The ciphertext will be twice the length of the plaintext.

Porta Cipher

class pycipher.**Porta**(*key='FORTIFICATION'*)
The Porta Cipher is a polyalphabetic substitution cipher, and has a key consisting of a word e.g. 'FORTIFICATION'.

Parameters **key** – The keyword, any word or phrase will do. Must consist of alphabetical characters only, no punctuation of numbers.

decipher(*string*)
Decipher string using Porta cipher according to initialised key. Punctuation and whitespace are removed from the input. For the Porta cipher, enciphering and deciphering are the same operation.

Example:

```
plaintext = Porta('HELLO').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The deciphered string.

encipher(*string*)
Encipher string using Porta cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Porta('HELLO').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

Railfence Cipher

class pycipher.**Railfence**(*key=5*)

The Railfence Cipher has a single number that forms the key. For more info on the Railfence cipher see http://www.practicalcryptography.com/ciphers/rail-fence-cipher/ .

Parameters **key** – an integer, must be greater than zero.

decipher(*string*, *keep_punct=False*)

Decipher string using Railfence cipher according to initialised key.

Example:

```
plaintext = Railfence(3).decipher(ciphertext)
```

Parameters

- **string** – The string to decipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The deciphered string.

encipher(*string*, *keep_punct=False*)

Encipher string using Railfence cipher according to initialised key.

Example:

```
ciphertext = Railfence(3).encipher(plaintext)
```

Parameters

- **string** – The string to encipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The enciphered string.

Rot13 Cipher

class pycipher.**Rot13**

The Rot13 Cipher has no key, it is commonly used just to hide text. This cipher encrypts a letter according to the following equation:

```
c = (p + 13)%26
```

where c is the ciphertext letter, p the plaintext letter. This is equivalent to the Caesar cipher with a key of 13. For more details on the rot13 cipher, see http://www.practicalcryptography.com/ciphers/rot13-cipher/ .

decipher(*string*, *keep_punct=False*)
Decipher string using rot13 cipher. The Deciphering and enciphering operations are identical.

Example:

```
plaintext = Rot13().decipher(ciphertext)
```

Parameters

- **string** – The string to decipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The deciphered string.

encipher(*string*, *keep_punct=False*)
Encipher string using rot13 cipher.

Example:

```
ciphertext = Rot13().encipher(plaintext)
```

Parameters

- **string** – The string to encipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The enciphered string.

Simple Substitution Cipher

class pycipher.**SimpleSubstitution**(*key='AJPCZWRLFBDKOTYUQGENHXMIVS'*)
The Simple Substitution Cipher has a key consisting of the letters A-Z jumbled up. e.g. 'AJPCZWRLFBDKOTYUQGENHXMIVS' This cipher encrypts a letter according to the following equation:

```
plaintext =  ABCDEFGHIJKLMNOPQRSTUVWXYZ
ciphertext = AJPCZWRLFBDKOTYUQGENHXMIVS
```

To convert a plaintext letter into ciphertext, read along the plaintext row until the desired letter is found, then substitute it with the letter below it. For more information see http://www.practicalcryptography.com/ciphers/simple-substitution-cipher/.

Parameters **key** – The key, a permutation of the 26 characters of the alphabet.

decipher(*string*, *keep_punct=False*)
Decipher string using Simple Substitution cipher according to initialised key.

Example:

```
plaintext = SimpleSubstitution('AJPCZWRLFBDKOTYUQGENHXMIVS').
↪decipher(ciphertext)
```

Parameters

- **string** – The string to decipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The deciphered string.

encipher(*string*, *keep_punct=False*)

Encipher string using Simple Substitution cipher according to initialised key.

Example:

```
ciphertext = SimpleSubstitution('AJPCZWRLFBDKOTYUQGENHXMIVS').
↪encipher(plaintext)
```

Parameters

- **string** – The string to encipher.
- **keep_punct** – if true, punctuation and spacing are retained. If false, it is all removed. Default is False.

Returns The enciphered string.

Vigenere Cipher

class pycipher.**Vigenere**(*key='fortification'*)

The Vigenere Cipher has a key consisting of a word e.g. 'FORTIFICATION'. This cipher encrypts a letter according to the Vigenere tableau, the algorithm can be seen e.g. http://practicalcryptography.com/ciphers/vigenere-gronsfeld-and-autokey-cipher/

Parameters **key** – The keyword, any word or phrase will do. Must consist of alphabetical characters only, no punctuation of numbers.

decipher(*string*)

Decipher string using Vigenere cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
plaintext = Vigenere('HELLO').decipher(ciphertext)
```

Parameters **string** – The string to decipher.

Returns The enciphered string.

encipher(*string*)

Encipher string using Vigenere cipher according to initialised key. Punctuation and whitespace are removed from the input.

Example:

```
ciphertext = Vigenere('HELLO').encipher(plaintext)
```

Parameters **string** – The string to encipher.

Returns The enciphered string.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

S

V