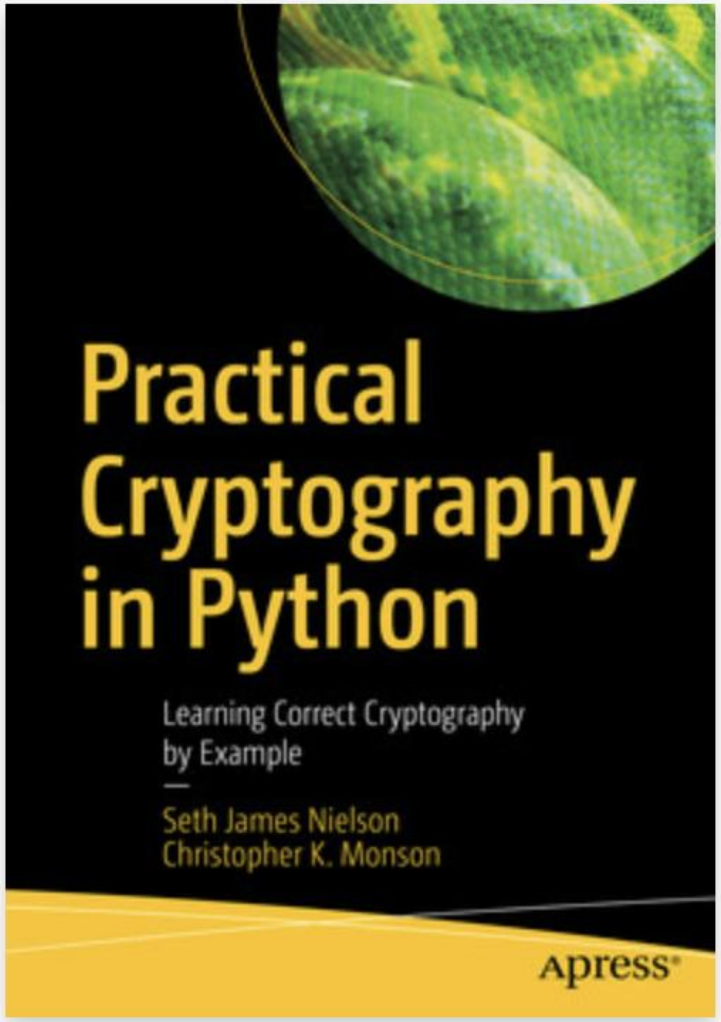
Week 11 – Basic operations

Recommended reading



The book is available to you via the library

Technology stack

- Python 3
  Link to a Python Cheat Sheet
- cryptography.io
  Link to the library

Technology stack

- Python3
- Cryptography.io
- Recommended reading: Chapter 1 from the book of "Practical Cryptography in Python"

Topics

- Conversions
- XORing
- Crack XORed's data
- Rotating ciphers (Caesars cipher)

String to/from Bytes

Assuming the following string

```
str1 = "Hello World!"
```

```
>>> print(str1)
Hello World!
>>> type(str1)
<class 'str'>
```

You can convert it to a bytes object using

bytes([source[, encoding[, errors]]]) e.g.

```
str1_bytes = bytes(str1, 'utf-8')
```

```
>>> print(str1_bytes)
b'Hello World!'
>>> type(str1_bytes)
<class 'bytes'>
```

And convert the bytes to a string:

```
str2 = str(str1_bytes, 'utf-8')
```

Bytes, bytearray

Byte objects are immutable

```
obj1 = bytes(2)
```

```
>>> obj1
b'\x00\x00'
>>> obj1[0] = 9
Traceback (most recent call
last):
  File "<stdin>", line 1, in
<module>
TypeError: 'bytes' object does
not support item assignment
```

If a mutable object is required use bytearrays instead

```
obj2 = bytearray(2)
```

```
>>> obj2
bytearray(b'\x00\x00')
>>> obj2[0] = 3
>>> obj2
bytearray(b'\x03\x00')
```

To decode a byte/bytearray to a string use the `.decode()` member function.

Bytes to/from hex

Assume the byte literal

```
obj1 = b”Hello World!”
```

We can convert it to its hexadecimal value as follows

```
hex_obj1 = obj1.hex()
```

```
>>> hex_obj1
'48656c6c6f20576f726c6421'
>>> type(hex_obj1)
<class 'str'>
```

We can covert a hexadecimal value to a byte as follows

```
obj2 = bytes.fromhex(hex_obj1)
```

```
>>> obj2
b'Hello World!'
>>> type(obj2)
<class 'bytes'>
```

Other conversions

Convert a hexadecimal value to an integer

```
int1 = int(hex_obj1, 16)
```

```
>>> int1
2240553423075392865078164790 5
```

And for reverting it to a hex

```
hex_int1 = hex(int1)[2:]
```

```
>>> hex_int1
'48656c6c6f20576f726c6421'
>>> type(hex_int1)
<class 'str'>
```

Interesting fact: The `int` type in Python3 is unbounded!

Other conversions (2)

Convert an integer to a binary

```
bin1 = bin(100)
```

```
>>> bin1
'0b1100100'
```

And for reverting it to an integer

```
int1 = int(bin1, 2)
```

```
>>> int1
100
```

XORing…

The XOR operator in Python is ^

Assume 2 integers, e.g. 10 and 7

What is the value of 10 ^ 7 and why?

XORing (2)

Let's assume we want to XOR the following strings

"a" with "b"

How can we do this?

What is the result?

XORing (2) - solution

Assuming the string literals ”a” and “b”

```
>>> a = b"a"
>>> b = b”b”
>>> ha = a.hex()
>>> ia = int(ha, 16)
97
```

Similarly, it’s 98 for “b”. And 97 ^ 98 = 3

Or we could XOR the bytes directly

```
>>> print(a[0]^b[0])
3
```

Structure of your code…

Modules you want to import

```
import XYZ
```

List of functions you implement

```
def  myFunction():
        # TODO

        return # TODO
```

Have a main section to call your functions

```
If __name__ == “__main__”:
        x = myFunction()
```

Task 1

- Write a function in Python3 that takes two hexadecimal values of equal length and returns their XOR value.

- To test your function , use the following strings:

- 49276d20746865206b657921020486f6f72726179 212020492063616e20656e637279707421
- 00534d571b1a0e534a452e444c4c680b001c17405 97648413d000241095200f17520e1f064a

Task 1

- You should get:
- 497420776f726b73212057656c6c20646f6e652120576861742061206772656174207 76f726b
- Decode the above hexadecimal considering the UTF-8 encoding. What does it say?

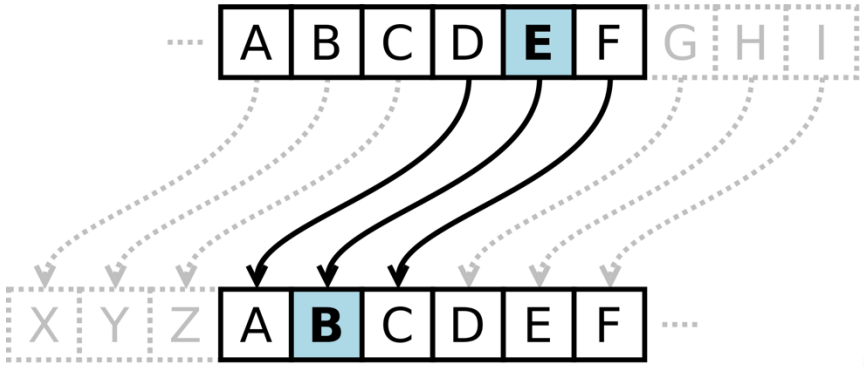
Task 2

- The following hex-encoded string is XOR'd against a single character (uppercase).
- 16203a6f242120386f3b272a6f242a366f212038636f3d2628273b70
- Find the key and the hidden message.

Task 3

- Implement the Caesar cipher (shift cipher). All letters of the alphabet should be supported, i.e. uppercase and lowercase
- Rotation: 10 PT: “Hello World” CT: "Rovvy gyBvn"
- A useful data structure to use: dictionaries

```
dict1 = {}
Dict1[“E”] = ”B”
```



Rotating ciphers

- What if your rotation is greater than your alphabet's size?
  - Modular arithmetic

$$\frac{\alpha}{\beta} = q \; remainder \; r$$

Where α: dividend, β: divisor, q: quotient, r: remainder

  - The modulo operator in Python is %

Behaviour of % with negative numbers

- The result depends on the programming language
- Python calculates the remainder as:

$$r = \alpha - (\beta * floor\left(\frac{\alpha}{\beta}\right))$$

Where floor is the `math.floor(x)` method which returns the floor of `x`, the largest integer less than or equal to `x`.

- What is the result for the following?
  - `math.floor(1.1) = …`
  - `math.floor(-1.1) = …`

Task 3

```
def build_tables(rotation_number):
        # TODO
        # The function should return 2 dictionaries that have the mapping
        # of a plain character to the encrypted one and vice versa,
          respectively
        return (plainToCipher, cipherToPlain)

def encrypt(plainText, plainToCipher):
        # TODO
        # The function should encrypt the provided plainText using
        # the plainToCipher dictionary built by function build_tables
        return ...
```

Task 3

```
def decrypt(cipherText, cipherToPlain):
        # TODO
        # The function should decrypt the provided cipherText using
        # the cipherToPlain dictionary built by function build_tables

        return ...
```