**Libraries to use: package Crypto** [**http://pythonhosted.org/pycrypto/**](http://pythonhosted.org/pycrypto/) **or cryptography** [**https://cryptography.io/en/latest/**](https://cryptography.io/en/latest/)

[**http://docs.python-guide.org/en/latest/scenarios/crypto/**](http://docs.python-guide.org/en/latest/scenarios/crypto/)

**All submissions will be on GitHub. The commit time stamp will be your submission time. No late submissions are allowed. Your last commit before the submission deadline will be graded.**

**Warm-up Exercises (Week 1)**

1. a) Write a program called *hex2Base64* to convert hex to base64. The hex string comes from the user.

Do not use any inbuilt functions for this conversion. Implement the conversion algorithm yourself. The program must display the base64 output on screen.

b) Write another program called *b64ConversionTester* that takes a text file as input, calls *hex2Base64* to convert it to base64 and then calls an inbuilt function (from a popular vetted library) to do the conversion and compares the result. The output must “True” or “False”, where True implies your output matched with the one generated by the system.

The input text file will contain hex strings (small letters), one on each line (user will provide the file name as an argument when calling the program). The hex strings may contain one or more whitespaces in between characters (e.g. ea b7 5 e a 8) and/or 0x as prefix. Your program should be able to handle these. If there are multiple lines in the file, your program should print True/False for each line.

Feel free to use look-up tables/dictionary structures as needed.

Clearly state the assumptions you make on the input (capital letters, length of input, etc.) and justify those assumptions (make a case why these are practical assumptions). For inputs that you think are not valid and may occur in the real world, create appropriate error handling.

Your submission must include a READ ME files for both 1a and 1b detailing the usage format and any other peculiarities of your implementation.

1. XOR: Implement a function that takes two equal length hex strings and XORs them. The output must also be in hex. Print the output to the screen.
2. The hex string given below has been XOR’d against a single character; find the key and decrypt the message. You can use English plaintext frequencies to rank the outputs.

TODO: Create a dummy encrypted input long enough that frequency analysis becomes possible.

**Week 2: Getting Serious**

1. General XOR cipher (Vigenere Cipher): Given a key ‘UNO’, encrypt the following text.

Your output should be:

TODO: Pick a text to encrypt

1. Breaking repeating-key XOR: Do not use any libraries that have inbuilt analysis algorithm

This will take time: One week for sure

**Week 3: Block Ciphers**

1. AES in ECB mode: give different encrypted text with key “NO PAIN NO GAIN!”

This takes maybe 30 minutes to 1 hour

1. Detect AES in ECB mode: Replicate Matasano’s file with different text and a twist

1-2 hours

1. Implement PKCS#7 padding for general block length. Give a random message and ask to pad to 160 bit block

I took the sentence “This is a Saturday” and the output was: 'This is a Saturday\x02\x02'

**Takes 5 minutes**

1. Implement CBC mode: Implement CBC mode by hand using the ECB function you wrote earlier. Make it a function in this program. It must encrypt and decrypt. And use the XOR function from the previous exercise to combine them.

**Week 4: Block Ciphers Continued**

1. PKCS#7 padding validation

This is needed later for CBC padding oracle attack

1. CBC bitflipping attacks

Will take one-week probably

**Week 5: Block Ciphers Continued**

1. The CBC padding oracle attack

Will take one-week for sure

1. Implement CTR mode

**Week 6: Stream Ciphers**

1. Implement Mersenne Twister RNG
2. Crack a MT19937 seed
3. Clone an MT19937 RNG from its output
4. Create the MT19937 stream cipher and break it

**Week 7: Stream ciphers continued**

1. Break "random access read/write" AES CTR
2. CTR bitflipping
3. Recover the key from CBC with IV=Key – find a real world example of this error for context

**Week 8: Hash functions**

1. *Implement* SHA-1 keyed MAC – write your own code
2. Break a SHA-1 keyed MAC using length extension
3. Break an MD4 keyed MAC using length extension

Week 9: