Homework 1 (Setting up cryptography and get familiarized to it)

We shall use Python to do all the homeworks. So, if you are not familiar with Python, it's a good time to get yourself acquainted with Python. To be specific, we shall use Python 2.7 (Why not 3.3? Because 2.7 is much more pythonic than 3.3.)

Install the following first.

1. First if you don't have Python 2.7, you have to install Python 2.7. This site might be a good place to start with. https://www.python.org/about/gettingstarted/. Remember the oracle named Google. It will be a great friend in the process of learning Python. You may want to look at http://docs.python-guide.org/en/latest/writing/style/ for better Python coding style guide.
2. Next best thing for working with Python is to install python package manager called pip. Follow the isntruction at https://pip.pypa.io/en/stable/installing/ to install pip. After this you can install most of the python packages just by running the command "pip install <package name>" command.
3. (Optional) Python Virtual Environment is a tool used to separate the python useful if you want to keep the dependencies of the python installation separate for different python projects. Check http://docs.python-guide.org/en/latest/dev/virtualenvs/ for more information.
4. Install Cryptography package. (https://cryptography.io/en/latest/). To test whether the installation is successful or not you can run the following code.
   (Taken from https://cryptography.io/en/latest/fernet/)

```
>>> from cryptography.fernet import Fernet
>>> key = Fernet.generate_key()
>>> f = Fernet(key)
>>> token = f.encrypt(b"my deep dark secret")
>>> token
'...'
>>> f.decrypt(token)
'my deep dark secret'
```

5. (Recommended) Install pytest. If you have installed pip, then simply run the following command.

   `$ pip install pytest` (you may need to run this with superuser privileges)

Next, look at the Layout section of the Cryptography library (https://cryptography.io/en/latest/#layout). The low-level cryptographic APIs are provided by the module called hazmat. The website has a warning about using this module in production without enough expertise:

> *"These are often dangerous and can be used incorrectly. They require making decisions and having an in-depth knowledge of the cryptographic concepts at work. Because of the potential danger in working at this level, this is referred to as the 'hazardous materials' or 'hazmat' layer."*

But, we shall learn how to use this securely and design user-level custom APIs during our assignments. You are more than welcome to look at the high-level API portion of the library (called `Fernet`), especially the source code. This will give you an idea of how to write high-level APIs and test cases, at least in the style of this library.

The assignment

The intention of the first assignment is to get you set up with the library and write your first encryption routine. Finish this by Feb 7, 2017 before the class. Late days can be used with subsequent assignments, but *not* this first one. We will elaborate on the late days policy shortly.

We shall create a simple stream cipher by using the AES block cipher in counter mode. To do so we will rely on the cryptography library's implementation of AES. Note, block cipher in counter mode is already implemented in the library. But, **please don't use the counter mode API provided with the library. Instead we shall build the stream cipher from the one block AES implementation.** There is no clean way to get access to the raw one block AES implementation. As a workaround, we shall set up an AES `encryptor` in ECB mode, and encrypt one block at a time using that. Below is a small code to start with. (This is a common way of accessing block ciphers in libraries.)

In the assignments we shall also focus on testing our modules as thoroughly as possible. Many security and functionality bugs can be stopped from propagating into production environment if we test the high level libraries well. We recommend using **pytest (**`$ pip install pytest`**)** to test the module. It's very simple and widely used. If you want to use something else that is okay too.

Starter code is available from github. Download AESCtr.py and test_AESCtr.py and address the TODOs.

```
#  To run the test, type the following command in the command prompt (Requires pytest library).

            $ py.test
```

Your submissions will be graded on a few different axes:

1) **Correctness** Did you correctly implement encryption and decryption? Does your code handle edge cases that could occur? (50%)
2) **Code quality** Is your code clean and well-commented? Could another programmer take your code, read it, and understand what it is doing? Will your code produce informative and helpful error messages when a failure occurs? (25%)
3) **Test quality** Do your tests cover a substantial fraction of use cases? (25%)
4) **Extra points** If you're feeling ambitious and would like some extra credit, think about the following question: How do the design choices we make about the nonce for CTR mode (e.g. its length or how it is generated) affect its security? Do different choices make an implementation more or less suited to different settings? If so, which ones? Write it in a clearly-indicated comment in one of your source files.

Deliverables: Implement CTR mode by filling in the template from Github. Submit AESCtr.py with your CTR mode implementation and test_AESCtr.py with your tests. You should not modify the interface given in the template. Turn the two Python source files into CMS.