# Cryptography Homework 3c

This time we will save our ciphertext, nonce, and tag to a file using JavaScript Object Notation, or JSON. In the scripting I've done, the most vexing thing has been conversion between data types, so you might as well get some practice.

## JSON

Currently JSON is the most popular data structure for storing and transmitting data in plain text, because it is simple, fast, and easy to code. <https://www.oracle.com/database/what-is-json/> It is widely used in program APIs for data input, as a format for programs to output data, as well as for use in log files.

Please read <https://www.digitalocean.com/community/tutorials/an-introduction-to-json>

This is a sample of JSON data from that link:

{"users": [

{"username" : "SammyShark", "location" : "Indian Ocean"},

{"username" : "JesseOctopus", "location" : "Pacific Ocean"},

{"username" : "DrewSquid", "location" : "Atlantic Ocean"},

{"username" : "JamieMantisShrimp", "location" : "Pacific Ocean"}

] }

In this case, 'users' is an array of user data, containing the username and location for each user. It looks very much like a Python dictionary.

Our JSON data will be simpler than the example above. It will look like this:

{

"nonce" : < nonce data>,

"tag" : < tag data>,

"ciphertext" : < ciphertext data>

}

We have a problem, however. The JSON format does not allow for binary data, only string data, which complicates matters. We will encode our data using Base64. However, we have to be careful to match the data types of the inputs and outputs of AES, Base64, and JSON.

* Output of AES encryption is binary
* Input of Base64 encoding is binary (good, it matches AES output)
* Output of Base64 encoding is bytes
* JSON requires strings, not bytes, so we have to convert bytes to string

To use JSON, and to encode the AES binary output in Base64, we will need to import Python3's modules:  
import json  
import base64

To encode our binary data as a base64 string we will need to use statements like this:  
ciphertext\_b64 = base64.b64encode(ciphertext).decode()  
Links: [Encode](https://www.tutorialspoint.com/python/string_encode.htm) [Decode](https://www.geeksforgeeks.org/python-strings-decode-method/) [Base64](https://www.educative.io/answers/how-to-use-base64b64encode-in-python)

The first part to be executed is base64.b64encode(ciphertext), which puts the binary data in ciphertext into Base64, in the Python3 data type "bytes". The next part, .decode() , converts from the bytes data type to the string data type.

Once we have the data in Base64 strings, we will put it in a Python3 dictionary. Dictionaries can be specified in one line:  
message = {"nonce" : nonce\_b64, "tag" : tag\_b64, "ciphertext" : ciphertext\_b64}

or in multiple lines for easier reading:  
message = {"nonce" : nonce\_b64,  
 "tag" : tag\_b64,  
 "ciphertext" : ciphertext\_b64}

[Python Dictionaries](https://www.w3schools.com/python/python_dictionaries.asp)

Then we can use the JSON dump() method to save the data to a file:  
with open("encrypted.json", "wt") as file\_out:

json.dump(message, file\_out)

Links: [Python JSON](https://www.w3schools.com/python/python_json.asp) [json.dump](https://www.geeksforgeeks.org/json-dump-in-python/)

## Code for Encryption

Save your file for Lab3b encryption with a new name, so we don't overwrite it. Insert your new code into the new file as follows:

A screenshot of a computer program

Description automatically generated

## Code for decryption.

This is like the encryption code, but in the reverse order. Save a copy of your Lab3b decryption code under a new name. Remember to import the json and base64 modules.

Use the json.load() method to read the data from the file your encryption code created.  
with open('encrypted.json', 'rt') as file\_in:

message = json.load(file\_in)

This will store your data in a dictionary, identical to what was in message in the encryption code.

Then convert nonce, tag, and ciphertext from the dictionary into binary data:  
nonce = base64.b64decode((message["nonce"].encode()))

The first thing executed is message["nonce"] which grabs the data (value) with the key "nonce" from the message dictionary.

We stored the nonce as a Base64 string, but the base64 module requires its input to be in bytes. Then next piece converts the string to bytes (message["nonce"].encode())

Finally, the last part changes the Base64 bytes into the binary data that AES requires.  
nonce = base64.b64decode((message["nonce"].encode()))

The code from your Lab3b decryption used this line to create the object:  
cipher = AES.new(key, AES.MODE\_EAX, nonce)  
Note that it requires the nonce! You have to extract the nonce from your file **before** you call AES.new().

Now that you have extracted nonce, tag, and ciphertext from your file, and created the AES object, you can proceed as before. Use the AES.decrypt\_and\_verify() method to decrypt the file.

## An Aside for More JSON

The site <https://jqlang.github.io/jq/tutorial/> has excellent demonstration of the tool jq. If you need to extract useable information from large JSON files, jq is a great tool. Install jq in Linux using:  
sudo apt install jq

Execute the commands in the JSON tutorial in your Linux.

Note: If you insist on using Windows, replace   
curl 'https://api.github.com/repos/jqlang/jq/commits?per\_page=5'  
with  
(curl 'https://api.github.com/repos/jqlang/jq/commits?per\_page=5').content  
PowerShell sees curl as an alias to Invoke-WebRequest, which just outputs the beginning of long data strings. The content method causes it to display everything. You will also need to copy jq.exe from Canvas to your home directory.

The data you get for the first step is a large block of hard to read "stuff." It is the data that was logged for five commits to a GitHub server. Raw JSON isn't easy to read without a parsing tool like jq.

You will see that you can use jq to extract parts of the data for one commit, or from all five commits.