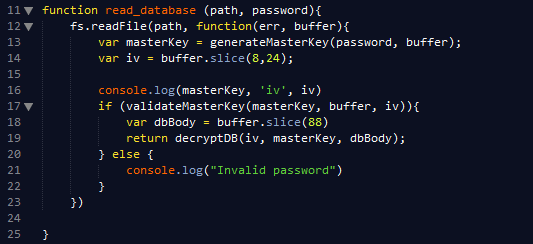
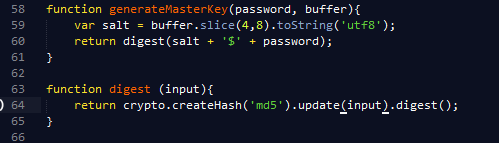
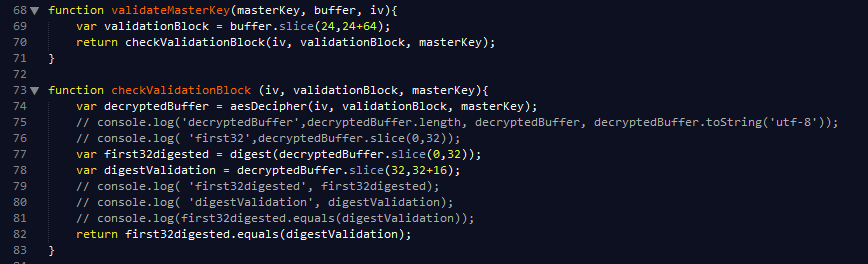
**How I Approached the Uber Challenge:**

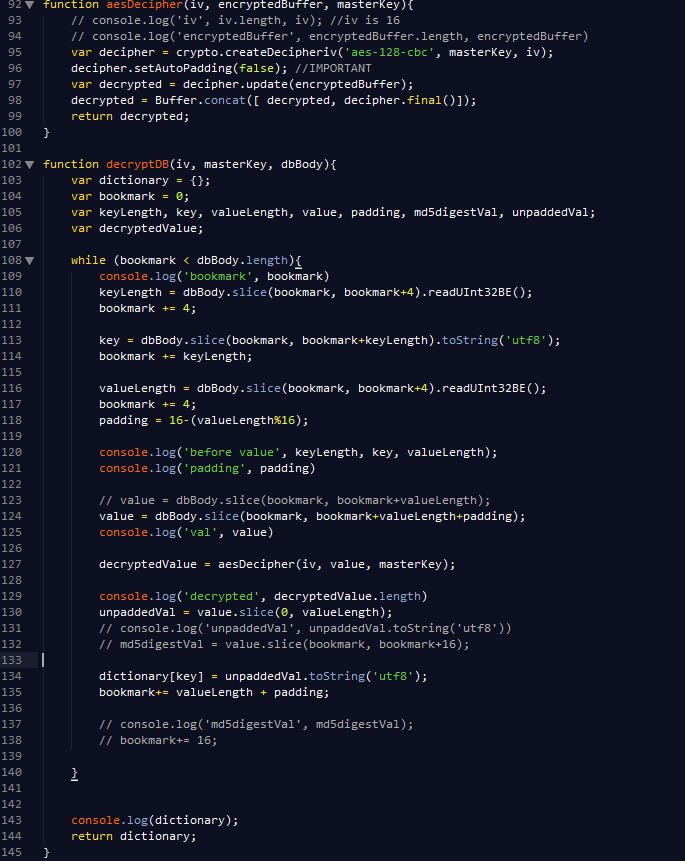
Read\_Database function will read the ‘demo.db’, passing in ‘uber’ as the password



* Fs.readFile() allows us to read the demo.db and returns its buffer representation.
* generateMasterKey() will get the salt and pass **‘wY10$uberpass’** to md5 digest to return a masterKey of ‘**8a 15 15 8a d6 4f aa f7 e7 16 b0 b7 17 fb eb 64**’ in hexadecimal.



* The IV is simply extracted from the demo.db **‘d2 5a 87 2e 9d 9a cb a3 42 e1 1f a9 25 5f 60 8e’** in hexadecimal.
* validateMasterKey() will decrypt the next 32 bytes of demo.db using AES-128-cbc and pass the decrypted value through the MD5 digest. This digest will be compared to the next 16 bytes, the digestValidation, and return a Boolean.
* if the MasterKey has been validated, take the dbBody and pass it into decryptDB() to parse out the keyLength, key, valueLength, and encrypted value block.



Based on the team’s feedback,

*"On write, the value is first serialized as JSON, then padded using padding algorithm, then encrypted."*

the encrypted block will be the size of the serialized JSON value plus the size of the padding. When decrypted, it should return the decrypted JSON and the padding.

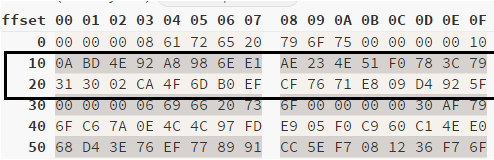
The screenshot of the buffer below is the part of the database where key-values are stored. I attempt to parse this buffer and got the following values:

* keyLength: **8**
* key: **‘are you’**
* valueLength : **‘16’**

so my padding is calculated to be **16.**

Since AES-128 encryption results in an output that’s the same size as its input, the encrypted block should be the size of the valueLength plus the padding – **32 bytes**.

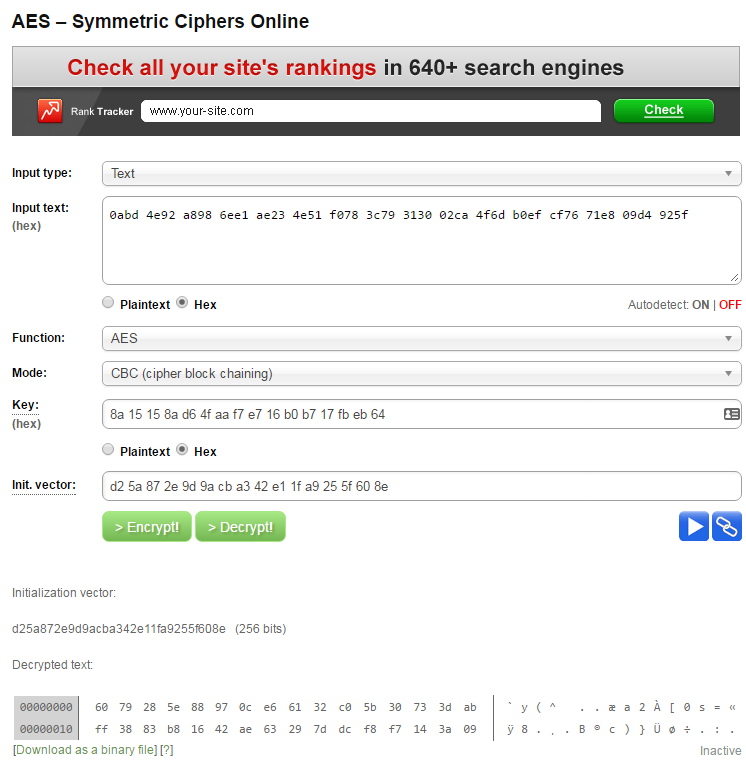
I passed the 32 byte encrypted block "0A BD 4E  .... D4 92 54" into aes-128-cbc (Line 124-127)



And resulted in this buffer:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 60 | 79 | 28 | 5e | 88 | 97 | 0c | e6 | 61 | 32 | c0 | 5b | 30 | 73 | 3d | ab |
| ff | 38 | 83 | b8 | 16 | 42 | ae | 63 | 29 | 7d | dc | f8 | f7 | 14 | 3a | 09 |

What I expect to see is that the second line should be a 16 byte padding of ‘10’, but it’s not so. I also cross checked my work using an online aes-128 CBC decrypter with the same values and got the same result:



It’s been really fun working on this challenge, and I learned a lot. Despite not getting the expected results, I wanted to do a write-up to show you my thought-process and how I approached this problem. If you have any feedback or comments regarding my code, they are very much welcomed.

Thank you!