

**Spring 2020 TTU - Big Data Security (ISQS-5342-001)**

Submitted by Anurag Sharma

**PART – 1 Preparation**

In preparation section of the project, which was the project proposal part we submitted to files which are described as following.

1. The data file we selected to secure.

File name: **R11659082\_data.docx**

1. Document describing the Symmetric Cipher and Hash Function we chose

With links to their Python implementation documentation.

**Symmetric Ciphers**

Salsa20

<https://pycryptodome.readthedocs.io/en/latest/src/cipher/salsa20.html>

Salsa20 is an efficient stream symmetric cipher. And it is favorite in ransomware world because of speed. Key is 256bits by preference but can work with 128-bit keys.

AES (CBC mode)

<https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#cbc-mode>

CBC (Cipher blocker chaining) is a form of block cipher. It usually adds complexity to encrypt data because of ciphertext block dependency on all plaintext blocks. It is common in general use.

**Hash Functions**

SHA-256

<https://pycryptodome.readthedocs.io/en/latest/src/hash/sha256.html>

It is a hashing function that converts text into string of 256 bits. Now a days it is used in Bitcoin network. It improves security and privacy.

HMAC

<https://pycryptodome.readthedocs.io/en/latest/src/hash/hmac.html>

Hash based message authentication code is cryptographic hash function used for message authentication code calculation which also verify the authenticity and integrity of a message.

**Part – 2 Execution**

The Execution part is processed in 4 sub parts which include the following:

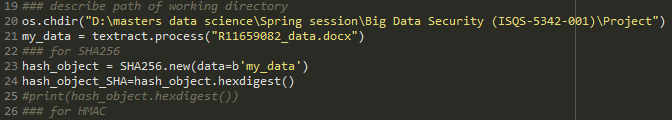
**Hash function:**

When we must map the data of arbitrary size to fixed size, we use hash function. The values are known as hash values or digests. When we apply hashing key to the data also known as hashing algorithm to get a hash value.

1. **Data Digest**

In this part the chosen data file is used and SHA-256 and HMAC hash functions are applied to generate hexadecimal data digest.

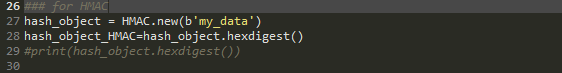
**Code for SHA256:**



Output:

**'e1f40338f64b974ab38d8711d51d8e848611aa8ca9f8d4dfdad0a61c30a77aac'**

**Code for HMAC:**

****

Output:

**'43b843d29673a9c9f490209c1d7a1904'**

Finally, at the end of the data digestion section we get two data digests generated from SHA256 and HMAC hash functions.

1. **Data Encryption**

In the data encryption part, we must apply both the symmetric ciphers to encrypt the selected data with same symmetric key.

**AES (CBC MODE) Symmetric Cipher:**

Key = b’1234567899876543’

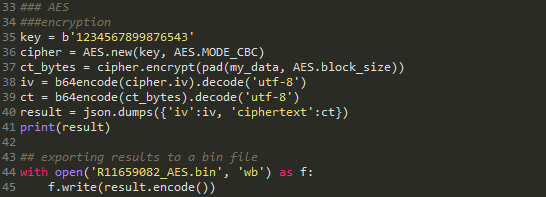
Exported file is **“R11659082\_AES.bin”**

Each plaintext block gets XOR-ed with the previous ciphertext block prior to encryption and create new object of CBC using algorithm.

It encrypts and decrypt data to have length of multiple block size.

Attribute iv is a read only attribute which hold bytes.

**Code for Encryption using AES (CBC Mode):**



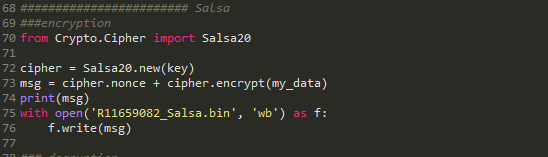
The encrypted result is stored in a .bin file.

**Salsa20 Symmetric Cipher:**

Key = b’1234567899876543’

Exported file is **“R11659082\_Salsa.bin”**

**Code for Encryption using Salsa20:**

****

The encrypted result is stored in a .bin file.

1. **Data Decryption**

In the data decryption part we import the respective bin files which we have stored in the directory as a result of encryption process from data encryption part and using the same key as we defined earlier we decrypt the data and get a plaintext as a new file and store as a docx file in the directory again.

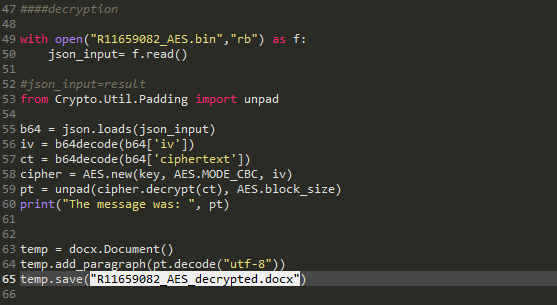
At the end of this process we will have two decrypted docx files.

**AES (CBC MODE) Symmetric Cipher:**

Key = b’1234567899876543’

Exported file is **"R11659082\_AES\_decrypted.docx"**

**Code for Decryption using AES (CBC Mode):**

****

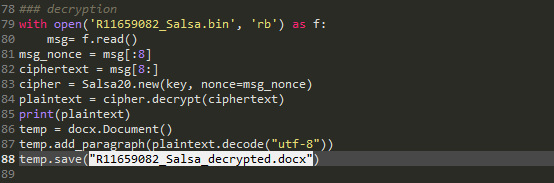
The decrypted result is stored in a .docx file.

**Salsa20 Symmetric Cipher:**

Key = b’1234567899876543’

Exported file is **"R11659082\_Salsa\_decrypted.docx"**

**Code for Decryption using Salsa20:**

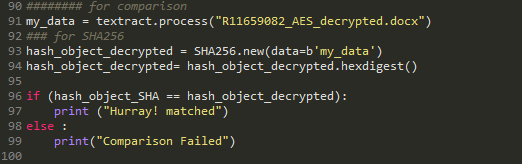
****

The decrypted result is stored in a .docx file.

1. **Data Integrity Verification**

In this part we again import the decrypted docx files to the system and again generate the hexadecimal data digest from the decrypted files. After generating the hexadecimal data digest for each decrypted file, we compare it with the hexadecimal data digest files from the Data digest part and verify either both the hash values are same.

**Code for SHA256 using AES\_decrypted:**



Output after Decryption:

**'e1f40338f64b974ab38d8711d51d8e848611aa8ca9f8d4dfdad0a61c30a77aac'**

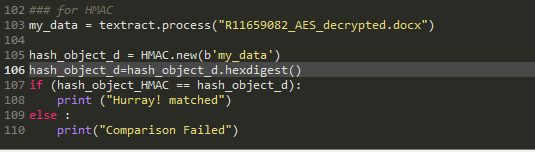
Output from Data Digestion:

**'e1f40338f64b974ab38d8711d51d8e848611aa8ca9f8d4dfdad0a61c30a77aac'**

Comparison Result:

**“Matched”,** it means that the encryption and decryption of data was successful and secure after using AES cipher.

**Code for HMAC using AES\_decrypted:**

****

Output after Decryption:

**'43b843d29673a9c9f490209c1d7a1904'**

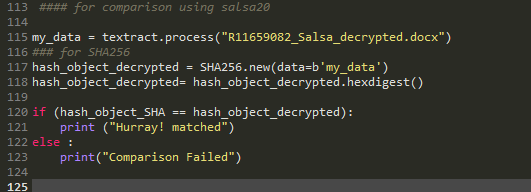
Output from Data Digestion:

**'43b843d29673a9c9f490209c1d7a1904'**

Comparison Result:

**“Matched”,** it means that the encryption and decryption of data was successful and secure using AES cipher.

**Code for SHA256 using Salsa\_decrypted:**

****

Output after Decryption:

**'e1f40338f64b974ab38d8711d51d8e848611aa8ca9f8d4dfdad0a61c30a77aac'**

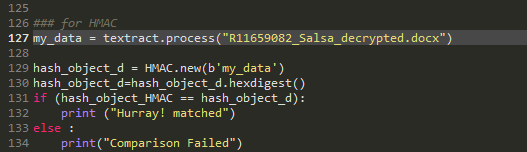
Output from Data Digestion:

**'e1f40338f64b974ab38d8711d51d8e848611aa8ca9f8d4dfdad0a61c30a77aac'**

Comparison Result:

**“Matched”,** it means that the encryption and decryption of data was successful and secure after using Salsa20 cipher.

**Code for HMAC using Salsa\_decrypted:**

****

Output after Decryption:

**'43b843d29673a9c9f490209c1d7a1904'**

Output from Data Digestion:

**'43b843d29673a9c9f490209c1d7a1904'**

Comparison Result:

**“Matched”,** it means that the encryption and decryption of data was successful and secure using Salsa20 cipher.