Sami Noor Syed
Cyber Security
OSU Fall 2023
10/18/2023

Programming Project 1

3.1-3.3 and 3.5-3.7 & Tasks 3.4 and 3.8

3.1:

Observation: this seems about right, each of the encrypted lines is at least as long as the original message if not longer

Plaintext: “Well here's some text that I need to encrypt... sorry it's not some clever quote :/”

Ciphertext (cleared formatting):

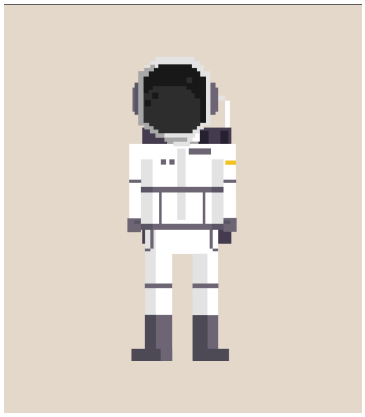
1) Aes-128-cbc:
    a) ��}�ʧ/��l|q��8�
    ���+�'sP^�*.y�q���i�����P5�~$gwᵾ��m
    Jl"!E^ʅu�f<\-�<���-�´����Z��
2) Aria-128-cfb:
    a) �w��>�*i�|�-��@�_
    ~�8�3=6�:�D�좟zq�;���5��`r���G;4"��H�l��
    ,'uA W"Bl.
3) Aes-128-ecb:
    a) ���W\��,T[��8"ljo`���T?
    ;q@H#ĸXfl*l��Λ��,��!ÎM�Aʷun!
    1��"�����^M��

3.2:

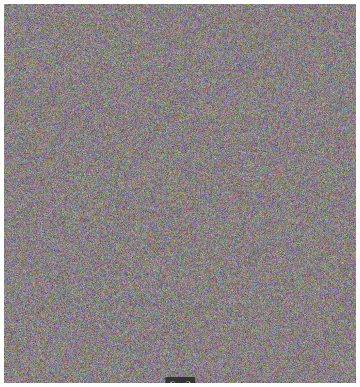
Observation: I really enjoyed the ECB image in this one! While CBC completely scrambles the image, in ECB the image is very easy to make out. This is because it uses the same algorithm on uniformly sized blocks of information from the original code. While we may not know what the original colors were, we can definitely make out the shape of the data as we have uniform encryptions on blocks of information much smaller than the original data set.

Pictures on the next page:

Original:



CBC:



ECB:

Image attribution: "https://www.freepik.com/free-vector/astronaut_2921422.htm#query=bmp&position=10&from_view=keyword&track=sph" on Freepik

3.3:

1) The contents of encrypt 1 and 2 match as they should. To be able to decrypt these messages, there has to be consistency in the way that messages are encrypted. If we encrypt a file twice using the same key and the same initialization vector, then we need to be able to decrypt that file with the same information to get the same result
2) It makes sense that using a different initialization vector would result in a different encrypted message especially since chaining block ciphering involves performing an xor with the initial plaintext. The resulting ciphertext is then xor'ed with the next bit of plaintext to produce new cipher text. In this way, a different initialization vector should change your entire cipher text.

3.5:

Using computed hashes seems like an interesting methodology of letting end receivers know that their message may or may not have been tampered with. The hashes all seem shorter than the original message, but that may be an artifact of the three algorithms that I used.

PlainText:
And we have another plaintext file, but this time we're using hashes to encrypt... right?

MD5 :
bdbae5750e5d8aae442a29aa21be7a15

Sha-256:
fe9e2fb6969106a62941531c414cdd80eac70f41f1a7cf2ec60cf43869b9860a

Sha-1:
3367ad397869dfb20b98273c8697886da6d31cf1

3.6:

1) It seems that the key size can be any length with HMAC, but with a little online snooping, I found out that Microsoft recommends that it should be 64 bytes long for the sha256 presumably to increase the resilience of the key in the face of attacks. I believe the key size depends on which cryptographic algorithm you use and what your individual security requirements are. Microsoft outlines later in the below link that a key that matches or exceeds the output size of the underlying hash function is often appropriate. Using a longer key provides extra security. (https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha256.-ctor?view=net-7.0)

3.7:

Plain.txt:
the cow jumps over the moon

1) Looking at the below hash codes with your eyes does not reveal much of a pattern, but if you look more closely at the binaries of those files, you see that when only one bit is changed, about 30-40% (unless I'm seeing patterns that aren't there) are the same from each of the single bit changes. When comparing the binaries of each of the h2's, this is far less as relative to each other, there are two differing bits. As we read in the explorations when using hashing algorithm like sha256 or 512, there is a waterfall or cascading effect which causes 50% to 80% of the other bits to change if there is only a one bit difference from the original message. In the second row of my table, I flipped all 4 bits, this resulted in a very different binary file from the original hash code. As more bits shift from the original message the resulting hashcode more drastically changes.

Bit flipped	Sha256	Sha 512
h1	6754b1f509d0e77c139aca3c12e8fa7ecd124564c87395b9744c21ed846798e0	af2bdc8845585202a615e4c4bdd90640e18578605ac1e33f8514b1bb3bb930db5fd812ad1b1c098837de7ed8cfeae7130440ffa46a8af8a737f764c7b56ded8b
H2 (after flipping bits 1, 49, 73 and 113)	114fd87f22612264c6d8f4491c18bbcab9bf6b09385c3959956ab99dc5d19121	098833cc088309fae231615905b94b7cfb2e3fc1a49b00311e4a113a4f44230cd53a95fe9e9d2fcb78f86a60e966ea197820df4bd11d67a1805b1f33f128880a
H2 bit 1	f47d526da017634b0ce746dbb643dc48ad0cc838eecca8f64520d1e3c9fa2679	6065de42ab15d8cd7768705aefd5db362bc78fdb23f9b1786def7dea0903497c672c60cf2d24f8de71604e68f0ae26ea1ed5b9171edd3daac7392d9d354e89f1
H2 bit 49	b74ce5761946a5e0ed436fb0b6788a007ade6cbb2dbc179d2c93b389c181f3	c1e58e8fc8e922007c0260fc2f123e1493a9b5b615bdb0ffa5b34fd0d711d52ac9abaf3c

	e4	ca7eb811867259585e63080ff50ba46b789a70a8938fa48bf0d071d5
H2 bit 73	633a942f760e3fdf4e263de043c57130d57c94b55c2f3d9671793a8620076c3d	618950fb69ed8fb7516baef2bb8ff70c99dd29cf12221f00f5370a720254c992c3c5b146a97103e10dc8ea2cf0d1432e483625ada8087434a29efb1b483830c4
H2 bit 113	5133126c0ba18f1e1e4e2970eb40d5dbfb8b1757610bb3c9658e431cbd6332ad	d8e811a92e94e8b415996bba3df53d58282f497b322566beb8ceee92cbab4ef7e9ddf822314a7f0d742442b786ae2ffc850d539312f29a781d4e3a122e940652