3.1

Plaintext:

<Document contains Plaintext>
92091385909123850981235819238123087591283 9

Plaintext (Hex)

3c446f63756d656e7420636f6e7461696e7320506c61696e746578743e0a
393230393133383539303931323338353039383132333538313932333831
323330383735393131323833390a

Plaintext (XXD Dump)

```
00000000: 3c44 6f63 756d 656e 7420 636f 6e74 6169  <Document contai
00000010: 6e73 2050 6c61 696e 7465 7874 3e0a 3932  ns Plaintext>.92
00000020: 3039 3133 3835 3930 3931 3233 3835 3039  0913859091238509
00000030: 3831 3233 3538 3139 3233 3831 3233 3038  8123581923812308
00000040: 3735 3931 3238 3339 0a                   75912839.
```

AES

-aes-128-cbc

8c145e1d1d035135dcb173d3bcacf7d9
d7838f88e0655d8dc10891ac76c51083
a2b66ae237c05d5c40dac4e8a5d68118
f8e53bb0d50ad57d3b01b8e84dae874d
239e8df035cebd1584197e13791dcfa2

-aes-128-cfb

bbc2e046b452dfbfdbbedc721b643d04
fd8ed1fbd583cd1be41af48d3aebafaf
3b5d8dadd9f79f7a16eb88bc264fef91
9b65fc35c6b056d14c524230e30bd4df
9a34a85b995ef2c8dd

-aes-128-ecb

768eaca6e8f58aada200d4634cc73eb0
8e11f8455a5c1f978efccfbd8e8c557e
5c030b6a3e02ad595c67874132a83215
30544721a729d006ef940c2ed5be57a5
3efb24dd4e958aa6ef23cd133a75b3a2

-aes-128-ofb

bbc2e046b452dfbfdbbedc721b643d04
5f27b957d73a2b9d833d7e886d6371ad
e5540d525d5066652c9759b605089991

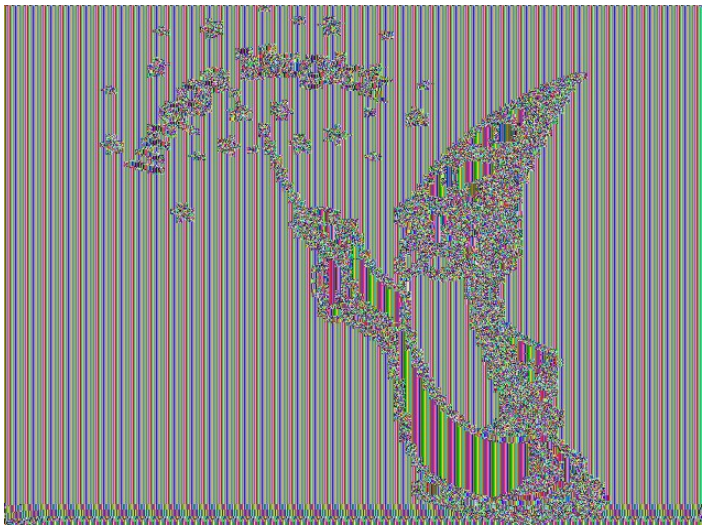
```
            361480a950aed41f0942592bed4d256c
            4a47e4e1ea5f76c16c

DES
      -des-cbc

            b628cbcb4172d0047617fe11db8f8239
            b357819918b1bd58199ed99cb6450d3c
            459500524fc855a57ef10ff5152d8f2c
            53f8077ed6278b38f66346c7ac804513
            537502f82710964d0ffca176c3adafa9

      -des-cfb

            d8409cbc6dc93675c1143f36b8b1e7ab
            2752fa9d23bc7977c29e486602b9594b
            03f551a5e96e34968ed5bccee4e32636
            417b83d9fb1260cc262e53ba747f64cb
            0d6226de75ff2d6ad2

      -des-ecb

            9e32769169796de1351c1abf9b4d5e45
            babfa5330cbda403393a9ad2057489ea
            5eb9b8445225cbeff880413dc19b3903
            305b89d0f4dbf20964ca29ddac252e73
            eb558f6df9a3e6a4d4a3d3dcf7612275

      -des-ofb

            d8409cbc6dc93675098013cf3d37f086
            abfd0dc88c0d75845c1dfc83a293caba
            18f3bf1c2ae9c4422b404307b6bff724
            d567c2c5c9d4e5fa1f6c5d3e58219f92
            19353dd5240a663981

SM4
      -sm4-cbc

            1a11139d074fdca43daecdd02d4d5a64
            73f7e2b3c5f3b7c4683cced179ae006e
            e29f8a00d1c5bea40f2283f1083cbdde
            5cec2984404fa38c72c10a21d623355a
            7ff0dd656e8ef675d75dbaa578e70b8b

      -sm4-cfb

            ba8dd2a708ab3a244a6d095f09f53711
            52e896b146d70733f072cb76d0b72a89
            0a6e585fb02cdc6be098250b69658aec
            095ecfa8401a663f4c43e3f4ae951105
```

```
        2fbb7129cf91ccf144

-smb-ecb

        c027e01627ffadd9fce8ecd6da73eb54
        129eb674410d662353ff1ae22f19c8dd
        af22a89f2e84c6082d503a278c16f479
        83b5d7c46b163052ef32e54707929ad6
        5be3e2e1e76cfbf5f84fe2cb9720d396

-sm4-ofb

        ba8dd2a708ab3a244a6d095f09f53711
        0399b7abe42d32cc3eb69c919e803db1
        920880fb6b49e8dfbe35d2e4138a9309
        49c37de17e8f7d5eb3aac187ce0dc1ea
        aa2e8da9fb4c71ac18
```

3.2

As I am offline as I am working on this, I chose the ImageMagick default image, shown below:

When encrypted with ecb encryption mode, the image retained quite a bit of resemblance to the original, and if you knew what text you were looking for, you could likely read it:



You can even still see the magick guy! Contrast this with the image when encrypted using cbc – in this case it was completely random.

Therefore, if I ever find myself using encryption to store images safely, an obvious choice is cbc over ecb.

3.3

I completed 4 different trials, keys {1:d8a9e0c2; 2:2392984023948} and iv's {1:111111111111111; 2:000000000000000}. These were done on a file containing "this contains INPUT DATA 'yeet'". Here are the results:

```
key 1, iv 1:
      bad8452042c44db57c97aaf31881cea9
      cc21b9ba4461e42adaa88502ab5c7eed
      f10a917c7025b3c5ebca45bcf46c7853
key 1, iv 2:
      9d78e6719cf7eb5ceb2b107a08a46f51
      88e5c655394edd5f7c22682bcb639e96
      58841b38a050c066689e300d259badaf
```

key 2, iv 1:
cb7a663065725cde2c93fa00c798623a
1e36f70f538cf0e049137efbb0537ede
876904a94fa7e19564f8fc86903f7ec9
key 2, iv 2:
2eccb9d7cd4a14c732eb6147af52b641
7af315a18a01af8e2bc74e2301d25f63
534f4bde141367369abe9a838d73b8a7

None of the contents match across the results using the same iv, or those that used the same key.

3.5

The message was a file containing the ascii string “this contains INPUT DATA 'yeet'”

SHA1(input.txt)= f4386834dd7117c040a7e9731396733b4012e2bd
SHA256(input.txt)= 82041e7ce5f33bb3f41dc8acd64fd043e2783feb0864cda67a608b4970e34402
SHA512(input.txt)=
890004428184a342250fcb71d89a94030a1c52f2295d7e69f03288770c73208203b1b7abe800dbcda22
d6ad69bb930db78ffcb6efc7fd8ea32328e96e0df4f05

3.6

No, HMAC does not require a certain length of key, as it ascertains that the key is of appropriate length by calculating K’ in the calculation below:

$$\mathrm{HMAC}(K, m) = \mathrm{H}\left(\left(K' \oplus opad\right) \parallel \mathrm{H}\left(\left(K' \oplus ipad\right) \parallel m\right)\right)$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

Therefore, if the key is too long to be used, then it gets it to length with a hash function. It only computes an Xor with the key.

3.7

SHA / BITFLIP AT	1	49	73	113
256	256	161	234	181
512	325	511	350	326

There appears to be no correlation between these numbers, except that (loosely) when SHA256 had many bits different in the output, SHA512 has not so many (relative to hash out size). The weakness of this trend indicates that there would likely be no change if more bits were flipped.