3.1

Plaintext:

Peter the piper picked a lot of pickles or something

```
10-197-144-147:Crypt jackson$ cat some.txt
Petter the pipper picked a lot of pickels or something
```

Ciphertext:

-aes-192-ofb:

00000000: f721 5896 66a7 ac94 e7a7 821d a4f0 f414 .!X.f.....
00000010: 969d 5ce2 9a1a fb8e 54a4 40ad e2f4 ac34 ..\.....T.@....4
00000020: cf5c da72 2e8f e699 351e ecc4 b820 58a9 .\.r....5.... X.
00000030: dcaa b589 03   .....

```
10-197-144-147:Crypt jackson$ openssl enc -aes-192-ofb -e -in some.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708 | xxd
00000000: f721 5896 66a7 ac94 e7a7 821d a4f0 f414  .!X.f.....
00000010: 969d 5ce2 9a1a fb8e 54a4 40ad e2f4 ac34  ..\.....T.@....4
00000020: cf5c da72 2e8f e699 351e ecc4 b820 58a9  .\.r....5.... X.
00000030: dcaa b589 03                             .....
```

-bf-cbc:

00000000: 3357 02fa 7802 f9f8 4b49 8c70 7672 448c 3W..x...KI.pvrD.
00000010: 7ac3 9ac0 1deb 9f42 d99b e649 aee7 0056 z......B...I...V
00000020: 9b00 e978 923d 1c27 2337 9f50 41f3 79b9 ...x.='#7.PA.y.
00000030: 6cc1 7f97 5a1e c7ab   l...Z...

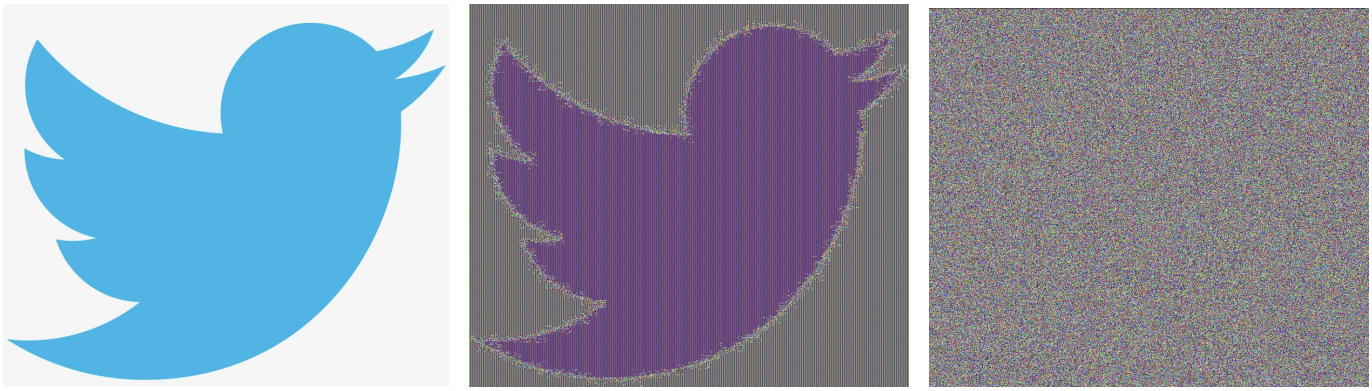
```
10-197-144-147:Crypt jackson$ openssl enc -bf-cbc -e -in some.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708 | xxd
00000000: 3357 02fa 7802 f9f8 4b49 8c70 7672 448c  3W..x...KI.pvrD.
00000010: 7ac3 9ac0 1deb 9f42 d99b e649 aee7 0056  z......B...I...V
00000020: 9b00 e978 923d 1c27 2337 9f50 41f3 79b9  ...x.='#7.PA.y.
00000030: 6cc1 7f97 5a1e c7ab                      l...Z...
```

-camellia-256-ecb:

00000000: 0c2a 9b3f cd03 03f0 e048 f4f5 45e1 2fa2 .*.?.....H..E./.
00000010: 4339 10cb be2b be2a 2280 de94 68e2 8ce3 C9...+.*"...h...
00000020: 3a33 fe79 92f1 29a6 fea1 ad3a 5bb7 1c37 :3.y..)....:[..7
00000030: c790 b0ae 414d 4203 0c22 e2de dcd3 cec4 ....AMB.."......

```
10-197-144-147:Crypt jackson$ openssl enc -camellia-256-ecb -e -in some.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708 | xxd
00000000: 0c2a 9b3f cd03 03f0 e048 f4f5 45e1 2fa2  .*.?.....H..E./.
00000010: 4339 10cb be2b be2a 2280 de94 68e2 8ce3  C9...+.*"...h...
00000020: 3a33 fe79 92f1 29a6 fea1 ad3a 5bb7 1c37  :3.y..)....:[..7
00000030: c790 b0ae 414d 4203 0c22 e2de dcd3 cec4  ....AMB.."......
```

3.2



The photos are as follow: Original Twitter logo picture, ecb encrypted, and cbc encrypted

The only other thing I got it to work with was my new film photos that I downloaded onto my computer. The top left is the original photo, top right is cbc encrypted and the final is ecb encrypted. It makes sense that these are much more fuzzy since the pixel values are much different in the clustered area than in the twitter logo picture. (Files are attached with the code zips)

3.3

The contents of encrypt1.bin and encrypt2.bin are the same. This is because the same algorithm was used along with the same key and initialization vector. Having the same initialization vector means that all the random numbers generated during encryption will be the

same. Furthermore, by using the same key each part of the plain text will be mapped to the same part of the ciphertext which will result in the same content.

The contents of encrypt1.bin and encrypt3.bin are not the same. This is because they use different initialization vectors. Initialization vectors are used so that the same string, even if it is encrypted using the same key, will not result in the same ciphertext. This adds a little more randomness and unpredictability to the cipher making it harder to break. In our case we can see that this is indeed the case as changing the iv changed the ciphertext.

encrypt1.bin: ^EE<85>9¼<9b><89>ÿ'^QE<89>È¿Ê¬ ¥^Uç^_^Y<8c>ù^OËÄMHzÖO
encrypt2.bin: ^EE<85>9¼<9b><89>ÿ'^QE<89>È¿Ê¬ ¥^Uç^_^Y<8c>ù^OËÄMHzÖO
encrypt3.bin: R£c<9d>Ý^_w<8b>¤:â£Ç°¨ý6~<84>ÿ<9d><85>û*P±%SÛt^V

3.5
Plaintext: Im like a biiiiiiiiird I only fly awaaaaaaaaaay

SHA1: 72ded9d2be92e072d29e1d3e7de9f7b5068fd881
MD5: ceeb7e04d8081711a0c7741fdaba4134
SHA512: ad402ca5b15772822d12de3237b88757c249a78502c8f56fe7b401962ea9a918

3.6
Within the inner workings of the HMAC algorithm a specific key size is needed which is the block size b. However, the algorithm handles the key size, so when executing a HMAC program, a specific key size is not needed. This is because if a key size that is too small is supplied it is padded with zeros, and if the key size is too big it is hashed down to a smaller size. **Therefore when using the HMAC algorithm like we are, a specific key size is not needed.**

"Definition of HMAC". HMAC: Keyed-Hashing for Message Authentication. sec. 2. doi:10.17487/RFC2104. RFC 2104.

3.7
Text in file: 'The cow jumps over the moon'

Hash SHA256: 29a0268117c4fa590198115bb3d796afde3839bb580d609e5c9c56bc6bcccfab

Hash SHA512:
3120a711f2c8affce2eb0253b28b7b802a0d985499c216d25e69a0d30128d549d090b98616a6fc9f998cca4f0d024ee68f1dcdec0afb6b615e7881f8de5bb3b2

Altered text file: ‘The!cow jumps over the moon’

Altered Hash SHA256:
4bfbf76e4d02603d5798c22d1eee66ce7aa4658469b0fe3d85ef881d417c959e

Altered Hash SHA512:
6962440c34375a47ca6a8f383c90adc0777fc132e6c8e3fb28f8d727cce92b22617cb09f6564b69
557952c896f5fe59ad196153a5d517de833f7a027ad0c108f

There was only 16 bits that were the same in SHA256 hash after flipping one of the bits in the text file (4 hex numbers are the same and 4 bits represent a hex so 4 * 4 = 16). There were also only 16 bits that were the same in SHA512 hash after flipping that same bit.

	SHA256 different bits	SHA512 different bits
Bit 1 flipped	232	468
Bit 49 flipped	252	476
Bit 73 flipped	236	492
Bit 113 flipped	236	496

The trend in the chart is that the biggest difference in the hash is produced when bits in the middle of the text file are flipped as opposed to the ones on the end. I tried flipping a bit used in representing the o in moon and it resulted in 236 and 492 different bits in the hash. I also flipped the last bit which resulted in 244 and 480 bits being different. When I flipped two bits in the text file the resulting hashes had 248 and 492 different bits in the resulting hashes. This does change the trend, as it seems that flipping multiple bits has a much bigger effect on the resulting hashes than their location in the string. When flipping all four of the bits on the table then the resulting hash has 244 and 476 different bits. This seems to show that regardless of which bits or how many bits are flipped the resulting hash will be very different. This corresponds to the avalanche effect which we know these hash algorithms have.

3.8

Weak	31737380	40392340	27030503	780527	293584	14779909
Strong	9136164	25383083	12146374	36643627		

Weak Collision: The weak collision averaged 21,253,240.60 trials to find a hash that matched the starting hash.

Strong Collision: The strong collision averaged 14,949,792.30 trials to find two hashes that were the same

The strong collision was the easiest to break using the brute force method. This, on average, took 6,000,000 less trials to find a collision than the weak collision. The reason for this has to do with the birthday paradox. Specifically the idea that it's much easier to find two people in a room with the same birthday than it is to find someone in a room with the same birthday as you. As mentioned in the lecture the probability for finding someone with the same birthday as you is

represented by the equation $p(n) = 1 - (364/365)^n$. This approaches 50% when n is 253. Compare this to the equation for finding two people with the same birthdays which is represented by the equation $p(n) = 1 - (365!\ /(365^n * (365 - n)!\ ))$ which approaches 50% when n is 23. Conceptually this makes sense because weak collision needs to find someone with the exact same birthday as you (1/365 for every person you see), which is harder than finding any birthday which two people share.