Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

Programming Project 1 - Encryption

3.1 Observation Task: Encryption using different ciphers and modes

Contents of *plaintext.txt*:

```
CS 370 - Intro to Security is a very interesting class!
```

First encryption command used (**aes-128-cbc**):

```
openssl enc -aes-128-cbc -e -in plaintext.txt -out cipher.bin \
  -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -p
```

Printed result:

```
hex string is too short, padding with zero bytes to length
salt=F3156FDD847F0000
key=00112233445566778889AABBCCDDEEFF
iv =01020304050607080000000000000000
```

Contents of *cipher.bin* (in hexadecimal using `xxd`):

```
00000000: 547e 4b29 0c23 bbaf dc11 9696 82cc d81a
T~K).#..........
00000010: 3843 acaf d06d 65c9 5e0c abcd 6856 f18b
8C...me.^...hV..
00000020: fbbc d60f 783a bba9 c44d 8b6d 2358 9186
....x:...M.m#X..
00000030: af72 3e26 f9a7 d10f ae48 fe04 e783 3a68
.r>&.....H....:h
```

Second encryption command used (**bf-ecb**):

```
openssl enc -bf-ecb -e -in plaintext.txt \
  -K 00112233445566778889aabbccddeeff \
  -iv 0102030405060708 | xxd
```

Printed result:

```
warning: iv not used by this cipher
00000000: c418 b38d 36a4 a073 9d1d e470 e6ee c099
....6..s...p....
00000010: 403b c127 e7b0 b169 5e98 b273 ed27 16dd
@;.'...i^..s.'..
00000020: 51ad 60a7 22b3 7b8f 13c3 b87b 63df 7722
Q.`.".{....{c.w"
00000030: 8f9f 9fac eaa5 ef93                       ........
```

Third encryption command (**cast5-cfb**):

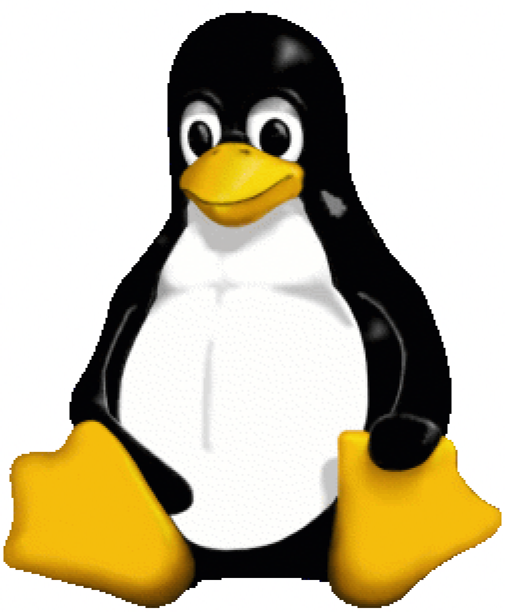
```
openssl enc -cast5-cfb -e -in plaintext.txt \
-K 00112233445566778889aabbccddeeff \
-iv 0102030405060708 | xxd
```

Printed result:

```
00000000: aea0 07bc bfd2 d825 ca11 4aa6 b128 269c
.......%..J..(&.
00000010: a5ca 91b3 bb24 9930 73b9 41aa b13b 66d5
.....$.0s.A..;f.
00000020: e19a be9f ad44 ed05 c835 b845 7b3c 142e
.....D...5.E{<..
00000030: b5cf d217 8f4e e3                        .....N.
```

3.2 Observation Task: Encryption Mode – ECB vs. CBC

Original Image 1:



Original Image 2:

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

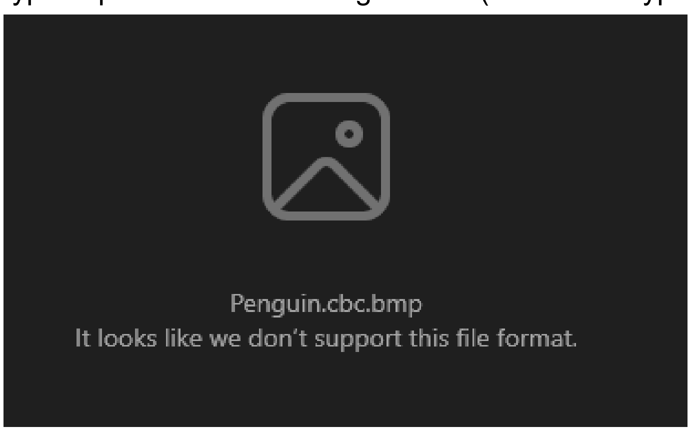
ECB encryption commands:

```
openssl des-ecb -e -in Penguin.bmp -out Penguin.ecb.bmp -K 1
openssl des-ecb -e -in TwitterLogo.bmp -out TwitterLogo.ecb.bmp -K
1
```

CBC encryption commands:

```
openssl aes-128-cbc -e -in Penguin.bmp -out Penguin.cbc.bmp \
   -K 1 -iv 1
openssl aes-128-cbc -e -in TwitterLogo.bmp -out TwitterLogo.cbc.bmp \
   -K 1 -iv 1
```

Encryption picture before editing header (all four encryptions give similar messages):



After copying the 54 byte bitmap headers from the original image files to the encrypted files, they can now be loaded in an image viewer.

Penguin.cbc.updated.bmp:

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

Penguin.ecb.updated.bmp:

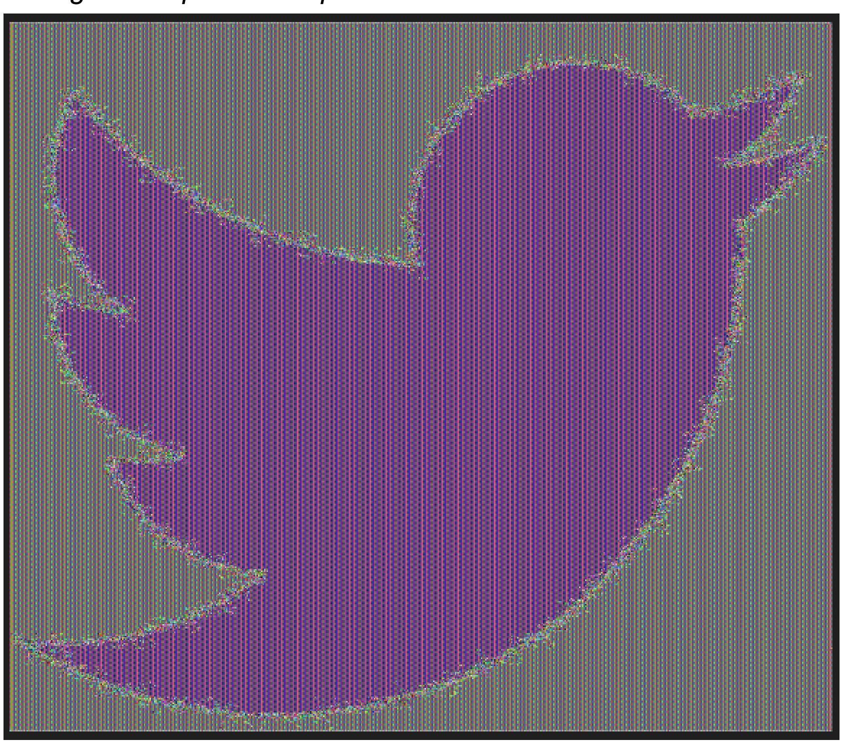
Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

The outline of the penguin is slightly visible.

TwitterLogo.cbc.updated.bmp:

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

TwitterLogo.ecb.updated.bmp:



Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

View of select bytes of *TwitterLogo.bmp* in Hex editor:

```
00000980 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 ÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷
00000990 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 ÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷
000009a0 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 ÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷
000009b0 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 ÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷
000009c0 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 ÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷÷
000009d0 f7 f7 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 ÷÷öööööööööööööö
000009e0 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 öööööööööööööööö
000009f0 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 öööööööööööööööö
00000a00 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 öööööööööööööööö
00000a10 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 f6 öööööööööööööööö
```

View of select bytes of *TwitterLogo.cbc.updated.bmp* in Hex editor:

```
00000980 00 30 92 f8 c6 95 89 bf d1 db 96 1a 9e 46 fa e7 .0’øÆ•‰¿ÑÛ–.žFúç
00000990 42 09 2a 14 70 b6 81 7a fe e5 50 a1 15 20 f0 5e B.*.p¶.zþåP¡. ð^
000009a0 83 18 b8 ea 46 04 c5 c1 c5 c2 82 f9 ff 2f d1 ff ƒ.¸êF.ÅÁÅÂ‚ùÿ/Ñÿ
000009b0 81 f0 52 9b 1d e3 17 7c 88 dc 07 6a 30 08 c1 dd .ðR›.ã.|ˆÜ.j0.ÁÝ
000009c0 5c 42 9b 18 55 1a ce b7 5c c6 9a 0a 59 32 dd 55 \B›.U.Î·\Æš.Y2ÝU
000009d0 dd 00 6f ff a7 6b b7 f2 9d f6 f6 39 92 86 c0 45 Ý.oÿ§k·ò.öö9’†ÀE
000009e0 e4 97 86 22 bd 55 65 4b d7 82 8c df b8 93 ac d1 ä—†"½UeK×‚Œß¸“¬Ñ
000009f0 63 42 3d 7d fa 16 70 22 fc ea 31 7b 45 c4 2d 40 cB=}ú.p"üê1{EÄ-@
00000a00 c9 02 ce 53 47 fd ec f5 39 5d 66 0e eb 9e e6 cf É.ÎSGýìõ9]f.ëžæÏ
00000a10 f5 1e 45 4e a2 1f 4f e5 25 67 89 a6 43 c5 5f b9 õ.EN¢.Oå%g‰¦CÅ_¹
```

View of select bytes of *TwitterLogo.ecb.updated.bmp* in Hex editor:

```
00000980 9e 9c fe 91 ef 82 8a 42 9e 9c fe 91 ef 82 8a 42 žœþ‘ï‚ŠBžœþ‘ï‚ŠB
00000990 9e 9c fe 91 ef 82 8a 42 9e 9c fe 91 ef 82 8a 42 žœþ‘ï‚ŠBžœþ‘ï‚ŠB
000009a0 9e 9c fe 91 ef 82 8a 42 9e 9c fe 91 ef 82 8a 42 žœþ‘ï‚ŠBžœþ‘ï‚ŠB
000009b0 9e 9c fe 91 ef 82 8a 42 9e 9c fe 91 ef 82 8a 42 žœþ‘ï‚ŠBžœþ‘ï‚ŠB
000009c0 9e 9c fe 91 ef 82 8a 42 9e 9c fe 91 ef 82 8a 42 žœþ‘ï‚ŠBžœþ‘ï‚ŠB
000009d0 84 24 fe 8d cd e8 39 58 59 f6 54 3f b7 4c 49 4c „$þ.Íè9XYöT?·LIL
000009e0 59 f6 54 3f b7 4c 49 4c 59 f6 54 3f b7 4c 49 4c YöT?·LILYöT?·LIL
000009f0 59 f6 54 3f b7 4c 49 4c 59 f6 54 3f b7 4c 49 4c YöT?·LILYöT?·LIL
00000a00 59 f6 54 3f b7 4c 49 4c 59 f6 54 3f b7 4c 49 4c YöT?·LILYöT?·LIL
00000a10 59 f6 54 3f b7 4c 49 4c 59 f6 54 3f b7 4c 49 4c YöT?·LILYöT?·LIL
```

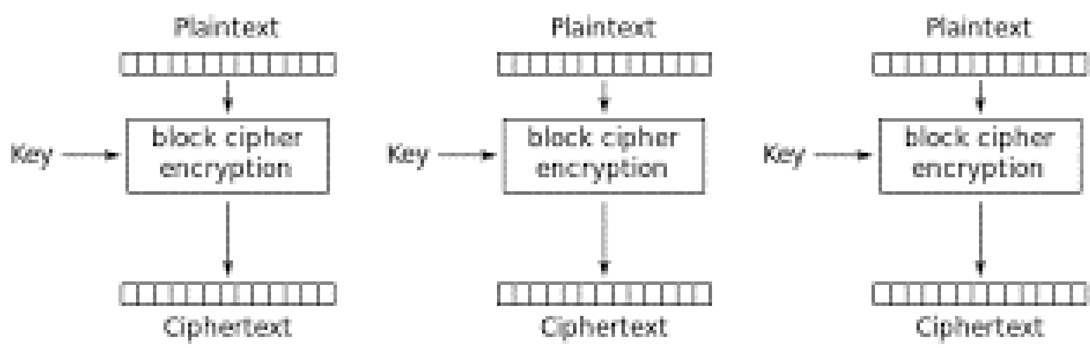
The difference between the two cypher methods is apparent.

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

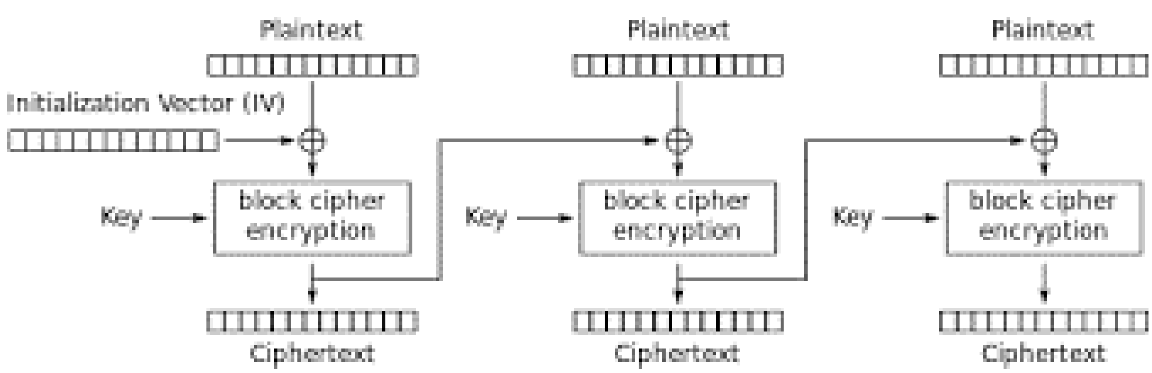
Explanation of interesting and surprising results:

EBC, a much simpler implementation than CBC, encrypts each block of plaintext independently. In an image like the Twitter logo above, large areas of blocks containing the same plaintext will therefore result in the same encryption, thus making the encrypted image still visible to the human eye.

CBC, on the other hand, uses the ciphertext of the previous block to inform the next block's encryption. This allows for a truly unique ciphertext for each block. The following diagrams show these differences in more detail. The diagrams were taken from [this Ubiq article by Eric Tobias]().



Electronic Codebook (ECB) mode encryption



Cipher Block Chaining (CBC) mode encryption

Note: The workflow for section 3.2 was inspired by [this Canvas page]().

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

3.3 Observation Task: CBC Encryption using different IVs

Contents of *plain.txt*:

```
Plain Jane vanilla bain.
```

First encryption command:

```
openssl enc -aes-128-cbc -e -in plain.txt -out encrypt1.bin \
 -K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

Contents of *encrypt1.bin* (in Hex using `xxd`):

```
00000000: 6245 3130 7c37 8c27 2488 16d2 9e1c c1c2
bE10|7.'$.......
00000010: 0aa3 b932 1975 b521 b314 3d88 a263 7092
...2.u.!..=..cp.
```

Second encryption command (using the same Key and IV):

```
openssl enc -aes-128-cbc -e -in plain.txt -out encrypt2.bin \
 -K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

Contents of *encrypt2.bin* (in Hex using `xxd`):

```
00000000: 6245 3130 7c37 8c27 2488 16d2 9e1c c1c2
bE10|7.'$.......
00000010: 0aa3 b932 1975 b521 b314 3d88 a263 7092
...2.u.!..=..cp.
```

Third encryption command (using the same Key but a different IV):

```
openssl enc -aes-128-cbc -e -in plain.txt -out encrypt3.bin \
 -K 00112233445566778899aabbccddeeff -iv 8877665544332211
```

Contents of *encrypt2.bin* (in Hex using `xxd`):

```
00000000: abbb fcf1 4d8f a36c ed04 e9d0 f94f 27c4
....M..l.....O'.
00000010: 9c7d b5a0 0e20 22b9 5296 5bea ee9e 1311  .}...
".R.[.....
```

The contents of *encrypt1.bin* and *encrypt2.bin* match because the exact same inputs were used on the same encryption algorithm, therefore yielding the same outputs. These outputs do not match that of *encrypt3.bin* because the IV (initialization vector) is different. If any of the inputs are changed in even the slightest manner, it will produce a different output.

3.4 Coding Task: Encrypting with OpenSSL

Please see attached files (*progProj1_3.4.py* and README.md).

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

3.5 Observation Task: Generating Message Digest and MAC

Contents of *plaintext.txt*:

```
Digestion is a process that converts nutrients in ingested food
into forms that can be absorbed by the gastrointestinal tract.
```

First command used:

```
openssl dgst -sha1 plaintext.txt
```

Output:

```
SHA1(plaintext.txt)= 329326d63729c6c52a1b006cd949b279b51b9ae2
```

Second command used:

```
openssl dgst -blake2b512  plaintext.txt
```

Output:

```
BLAKE2b512(plaintext.txt)=
c66d270733e96d1afcb13fab5a8f74a193c7dc7bd57812803c57d519ddeb813a87
398a0ae52806082727490909 6f5d45d7491b25fd859521c0b5ce3e1c80ca2f
```

Third command used:

```
openssl dgst -md5 plaintext.txt
```

Output:

```
MD5(plaintext.txt)= 94db4e530cc31e9b8896693f701bcba2
```

3.6 Observation Task: Keyed Hash and HMAC

Contents of *plaintext.txt*:

https://en.wikipedia.org/wiki/Shah

First HMAC-SHA256 command used:

```
openssl dgst -sha256 -hmac "1953" plaintext.txt
```

Result:

```
HMAC-SHA256(plaintext.txt)=
42df48a8c2bc64dadd95442e7b2e0a5ba7d6a4791b21e873ef27544176d0c451
```

First HMAC-SHA1 command used:

```
openssl dgst -sha1 -hmac "1953" plaintext.txt
```

Result:

```
HMAC-SHA1(plaintext.txt)= 23a7844feb3ade59f2fe301b0e902375f4969d88
```

Second HMAC-SHA256 command used:
```
openssl dgst -sha256 -hmac "1953consolidatedPower" plaintext.txt
```

Result:
```
HMAC-SHA256(plaintext.txt)=
43241884725aea5da9b5fef54b2919551e75525a57f436dbbf722326abf5066b
```

Second HMAC-SHA1 command used:
```
openssl dgst -sha1 -hmac "1953consolidatedPower" plaintext.txt
```

Result:
```
HMAC-SHA1(plaintext.txt)= 0343518ee43c7aeea5e68b59d909e7d05aac3c7e
```

Third HMAC-SHA256 command used:
```
openssl dgst -sha256 -hmac "" plaintext.txt
```

Result:
```
HMAC-SHA256(plaintext.txt)=
a9562eea2e551bb7840f4eb847e7ba6e7dff8345e2e91e169e68181f0b3243f5
```

Third HMAC-SHA1 command used:
```
openssl dgst -sha1 -hmac "" plaintext.txt
```

Result:
```
HMAC-SHA1(plaintext.txt)= c8c9a9caa0c71b5376618d797e39f35f76b9c145
```

As exemplified above, HMAC does not require a key of a fixed size. This is because it uses a cryptographic hash function, which allows for an input of any length to produce a fixed-length output. Notice in the examples above that the hashes of each function (sha1 and sha256) are of the same length (20 bytes for sha1 and 32 bytes for sha256).

3.7 Observation Task: The Randomness of One-way Hash

Content of *plaintext.txt*:
```
The cow jumps over the moon
```

SHA256 command:
```
sha256sum plaintext.txt
```

Result:
```
aa2f7b095a4e2fb3918fa20b2221544d61632aba9fd9e8afa353eaaefde776db
```

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

SHA512 command:

```
sha512sum plaintext.txt
```

Result:

```
c2e6c46087c682b62e51aa38d30b15146285f7960f45c5531af3967f170cbbb85b
0f7aac13f1e4b1eec2e640b4e8f28eb1e1f756f30f6bdc4c909fe31fe43229
```

Plaintext in Hex before any bit flipping:

plaintext2.txt

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	54	68	65	20	63	6f	77	20	6a	75	6d	70	73	20	6f	76	The cow jumps ov
00000010	65	72	20	74	68	65	20	6d	6f	6f	6e						er the moon

Plaintext in Hex before any bit flipping:

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	01010100	01101000	01100101	00100000	01100011	01101111	01110111	00100000	01101010	01110101	01101101	01110000	01110011	00100000	01101111	01110110	The cow jumps ov
00000010	01100101	01110010	00100000	01110100	01101000	01100101	00100000	01101101	01101111	01101111	01101110						er the moon

A copy of *plaintext.txt* was made to the file *plaintext2.txt* for experimentation with bit flipping.

Flipping the first bit and hashing yields the following results:

plaintext.txt plaintext2.txt

Address	0	1	Dump
00000000	11010100	01101000	Ôhe cow jumps ov
00000010	01100101	01110010	er the moon

SHA256:

```
241d3a76e3fe7f4ab8293f9499e9d890a0518db120b0bef7eb1a577c9af6bcac
```

Comparitor	0	1	1	1	0	0	0	1	0	0	0	1	0	0	0	1
Total Matching bits (256)	66						26%									

After the first two bits, the pattern of 1000 continues repeating.

SHA512:

```
64e4c29b1f87e9b193d1dac7dae4d1821264bfdd46607789f1c979903d8b9b566c
c0ceac0bedd017b0cc9a4693c5400534d1b416ca9784b7f04625704415298e
```

Comparitor	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Total Matching bits (512)	256						50%										

After the first three bits, the pattern of 1100 continues repeating.

Flipping the 49th bit and hashing yields the following results:

6	7	Dump
01110111	10100000	The cow.jumps ov
00100000	01101101	er the moon

SHA256:
`502a820be80154a503902da53791402a357a4b5f016b9cb7a18087beaf778b05`

Comparitor	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1
Total Matching bits (256)	126						49%									

After the 5th bit, the bits match with the repeating pattern 1010.

SHA512:
`880deb93243ef62dc09b103d8514859d87df08c202227f7e3abd0605e66a7422b`
`69d7fc61927579f74e741f63ca5e479324de4d5fd3414ed47bb4f9251f0eb12`

Comparitor	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1
Total Matching bits (512)	257						50%									

After the 3rd bit, the bits match with the repeating pattern 1010.

Flipping the 73rd bit and hashing yields the following results:

	9	a	Dump
0	01110101	11101101	The cow juips ov
1	01101111	01101110	er the moon

SHA256:
`3eb9cfab56313de7fc80a5a2f1abfcf6be8113d4014df9d3cbf6693e449101ea`

Comparitor	0	1	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1
Total Matching bits (256)	191						75%										

After the 3rd bit, the matching bits repeat with the pattern 0111.

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

SHA512
`dfabcd18df9716b96f5452f791a52bf324169d595d5d4d700aece28d63f15595a`
`46be84fc60a1006be394b0273be7cbbc166ab6f37dc510717b41542ac19f088`

Comparitor	1	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0
Total Matching bits (512)	130						25%									

After the 2nd bit, the matching bits repeat with the pattern 1000

Flipping the 113th bit yields the following results:

e	f	Dump
01101111	11110110	The cow jumps oö er the moon

SHA256:
`ca2d2499a3351c5a93947936a9b8903986983b60dda758ad4ac3f448a2e4934a`

Comparitor	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
Total Matching bits (256)	254						99%									

After the 3rd bit, all bits match.

SHA512:
`77f7edbee90ec1f23aab15a3ddcfa60c548e8e6f239d87fcc724881051752f2b1`
`e95f1dd4a24dec299e599635f6e6aa69a4388a3e708b8c12c92a5b96161caa6`

Comparitor	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0
Total Matching bits (512)	255						50%									

After the 3rd bit, the bits match in the pattern 0101.

Flipping bits 1, 49, 73, and 113 simultaneously yields the following results:

Dump
Ôhe cow.juips oö er the moon

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

SHA256:
03cf789d9a62add4f9e2e40a76e723e3c539e2fdd8dfb816754840991c9a7d6e

Comparitor	0	1	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0
Total Matching bits (256)	128	50%															

After the 5th bit, the matching bits repeat with the pattern 1100

SHA512:
30c5b80106847acbceb4de1a05fe3ad99702becb03eea57c3aa89480cf22a583b
e19a0069a516b385d8c1acb1be6564e5947522584336289840c5883a7ef46f6

Comparitor	0	0	0	0	1	1	0	1	1	1	0	1	1	1	0	1	1	1
Total Matching bits (512)	381	74%																

After the 6th bit, the matching bits repeat with the pattern 0111

Flipping only the 1st and 113th bit yields the following results:

Dump
Ôhe cow_jumps oö
er the moon

SHA256:
95d30961abdf59debdbbe9be1a09c9b9004359d2db90de70e7359f127d2c5e26

Comparitor	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total Matching bits (256)	2	1%															

After the 2nd bit, no bits match.

SHA512:
4610bd0e62918f16a4e1e6a4277769f726492bcbaf11809942147c352494c476f
9cb01e9bf92e39f379ffa857d7c867d8880c9b7c104e49057ce1460b11f5306

Comparitor	0	1	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
Total Matching bits (512)	384	75%															

After the 2nd bit, bits match in the repeating pattern 1110.

Flipping the entire first byte:

Seth Weiss
CS 370 - Intro to Security
Fall 2023
Programming Project 1 - Encryption

0	1	Dump
10101011	01101000	«he cow jumps ov
01100101	01110010	er the moon

SHA256:
a67f2e1a45b3eb564d9188fc2fff740471813e4ef6fe25a02dcb7502b4e4a035

Comparitor	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Total Matching bits (256)	130						51%									

After the 2nd bit, the matching bits repeat with the pattern 1100.

SHA512:
5acea7bda7e534dd399e9a982e0c257446a9cfbce560c3a41ea6fd9cb804479a5
dab5861d4400f8ddd07720c268e6602327ae605b77adb3f039cc52f61a61d00

Comparitor	0	1	1	0	0	1	1	1	0	1	1	1	0	1	1	1	0
Total Matching bits (512)	383						75%										

After the 4th bit, the matching bits repeat with the pattern 0111.

The following tables show the results of number of matching bits from changes in various bit indices and combinations.

bit index(es) flipped	1	49	73	113	1, 49, 73, and 113	bits 1 and 113	the entire first byte
Num matching bits SHA256	66	126	191	254	125	2	13
Num different bits SHA256	190	130	65	2	131	254	243
Percent similarity	26%	49%	75%	99%	49%	1%	5%
Num matching bits SHA512	256	257	130	255	381	384	383
Num different bits SHA512	256	255	382	257	131	128	129
Percent similarity	50%	50%	25%	50%	74%	75%	75%

bit index(es) flipped	1	1,2	1,2,3	1,2,3,4	1,2,3,4,5	1,2,3,4,5,6	1,2,3,4,5,6,7	1,2,3,4,5,6,8
Num matching bits SHA256	66	65	127	128	254	129	129	130
Num different bits SHA256	190	191	129	128	2	127	127	126
Percent similarity	26%	25%	50%	50%	99%	50%	50%	51%
Num matching bits SHA512	256	382	384	1	128	258	255	383
Num different bits SHA512	256	130	128	511	384	254	257	129
Percent similarity	50%	75%	75%	0%	25%	50%	50%	75%

bit index flipped	1	2	3	4	5	6	7	8
Num matching bits SHA256	66	65	128	65	128	191	126	127
Num different bits SHA256	190	191	128	191	128	65	130	129
Percent similarity	26%	25%	50%	25%	50%	75%	49%	50%
Num matching bits SHA512	256	255	384	254	256	129	128	3
Num different bits SHA512	256	257	128	258	256	383	384	509
Percent similarity	50%	50%	75%	50%	50%	25%	25%	1%

As can be seen in the tables above, a majority of the bit flip changes result in a 25%, 50%, or 75% match in bits between the hash created from the original text and those from the changed texts. There is no discernable pattern correlating the number of bits changed and the number of matching bits. A reason these percentages could be fairly consistent is in the fact that both SHA256 and SHA512 are designed with the avalanche effect in mind, which states that a single bit change to the input of a hashing algorithm should result in a significant change to the bits of the output. Having said that, when bits 1, 2, 3, 4, and 5 are simultaneously flipped, SHA256 produces a worrisome 99% matching bits. The same percentage of matching bits results from flipping bit 113 and hashing using SHA256. However, in hashing, there is no rounding or "close enough", so even a 99% similarity in bits will produce a vastly different hash. This is displayed by comparing the original hash with the hash of flipped bit 113.

Original SHA256 hash:

`aa2f7b095a4e2fb3918fa20b2221544d61632aba9fd9e8afa353eaaefde776db`

Flipped 113 bit SHA256 hash:

`ca2d2499a3351c5a93947936a9b8903986983b60dda758ad4ac3f448a2e4934a`

3.8 Coding Task: Weak versus Strong Collision Resistance Property

Please see attached files (progProj1_3.8.py and README.md).