Vincent Yasi

CS 370 -Intro to Security-

Programming Project 1

3.1

**Cipher 1**

Method: -aes-128-cbc

Plaintext: “Hello World”

Ciphertext (hex): 5361 6c74 6564 5f5f 1bbe 4e0a fc63 6f3e 9d33 f9ac bba2 142d 44be 02b3 95c8 88df

**Cipher 2**

Method: -aes-128-cfb

Plaintext: “Secret Message”

Ciphertext (hex): 5361 6c74 6564 5f5f a9cb 9633 cf68 815c adc8 7d28 5415 227f 57d6 218c b4b5

**Cipher 3**

Method: -bf-cbc

Plaintext: “Cryptography is fun!”

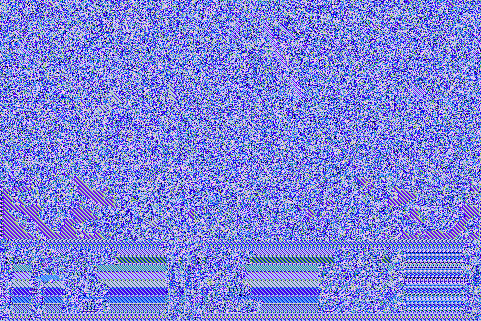
Ciphertext (hex): 5361 6c74 6564 5f5f b0fe f47a 6849 85b2 072e e4de 199e ec5b 8237 9b00 8975 3181 3521 c076 476c 5079

3.2

**Original**



**Encoded with ECB**



**Encoded with CBC**

A picture containing blue, court, people, player

Description automatically generated

I should have probably used a simpler picture for this section, but we can still see some patterns and “information leak” with the ECB picture. Toward the bottom where the battle menu is displayed (and the colors are mostly uniform) we can see the outline of the sections between the text. As well, a bit above that, we can also see patches where the clouds are rather uniform in the original picture. This all shows that some of the information of the original picture is still partially recognizable in the encrypted version for ECB.

The CBC version, however, shows none of these patterns, and overall looks like a bunch of random noise.

This demonstrates that CBC produces generally (pseudo)random ciphertexts, while ECB can create patterns in its ciphertexts, making them not truly random.

3.3

**Key Used:** 1234567890

**IV for Run 1 and 2:** 0987654321

**IV for Run 3:** 5555555555

My initial runs to produce encrypt1 and encrypt2 gave different ciphertexts, even though the inputs were the same. This is because I found out OpenSSL v.1.1.1 (the one I have installed) automatically salts encryptions. When I ran them again with the -nosalt flag, the two files were the same. This is because the two encryptions use the same key, IV, and plaintext. Therefore, because of the way CBC works, it gives the same output.

For the third run (with -nosalt used), the ciphertext was different than the first two. This is because, even though it uses the same key, the IV is different. With a different IV, the results of the first cycle will be different.

3.5

**Plaintext:** “Hello World”

**Hash Digest -sha1:** 0a4d55a8d778e5022fab701977c5d840bbc486d0

**Hash Digest -sha256:** a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e

**Hash Digest -sha512:** 2c74fd17edafd80e8447b0d46741ee243b7eb74dd2149a0ab1b9246fb30382f27e853d8585719e0e67cbda0daa8f51671064615d645ae27acb15bfb1447f459b

3.6

**Plaintext:** “Hello World”

**HMAC-SHA256 Key:** bongo

**HMAC-SHA256 MAC:** bd9e40f6733e868fa502a98109d42871818527d57fa2d5749f70af789b3345af

**HMAC-SHA1 Key:** key

**HMAC-SHA1 MAC:** cc24f1acdb06cf429bcf9861b6d708b6ec20a8fa

The answer to if the key needs to be a certain size is yes and no. HMAC uses blocks, so the key needs to be one blocklength in size. However, if the key is larger than a blocklength, then it can be trimmed down to size, and if it is shorter, than it can be padded to the correct length. So, the key should be a certain size, but if it is not, then it can easily be adjusted and keep the security of the hash.

3.7

**Plaintext:** “The cow jumps over the moon”

**Key:** bongo

**H1-SHA256:** aa2f7b095a4e2fb3918fa20b2221544d61632aba9fd9e8afa353eaaefde776db

**Binary:** 1010101000101111011110110000100101011010010011100010111110110011100100011000111110100010000010110010001000100001010101000100110101100001011000110010101010111010100111111101100111101000101011111010001101010011111010101010111011111101111001110111011011011011

**H2(1 flip)-SHA256:** d6708757feafa410bcdd428ae0311c7c4dd3c79970ed0962d166f130d516bf1e

**Binary:** 1101011001110000100001110101011111111110101011111010010000010000101111001101110101000010100010101110000000110001000111000111110001001101110100111100011110011001011100001110110100001001011000101101000101100110111100010011000011010101000101101011111100011110

**H2(1, 49, 73, and 113 flip)-SHA256:** 14727c57d4285b442a51be83c45048ab1d8b9fb6c60cfacc7be35d1d547d6adb

**Binary:** 0001010001110010011111000101011111010100001010000101101101000100001010100101000110111110100000111100010001010000010010001010101100011101100010111001111110110110110001100000110011111010110011000111101111100011010111010001110101010100011111010110101011011011

**H1-SHA512:** c2e6c46087c682b62e51aa38d30b15146285f7960f45c5531af3967f170cbbb85b0f7aac13f1e4b1eec2e640b4e8f28eb1e1f756f30f6bdc4c909fe31fe43229

**Binary:** 11000010111001101100010001100000100001111100011010000010101101100010111001010001101010100011100011010011000010110001010100010100011000101000010111110111100101100000111101000101110001010101001100011010111100111001011001111111000101110000110010111011101110000101101100001111011110101010110000010011111100011110010010110001111011101100001011100110010000001011010011101000111100101000111010110001111000011111011101010110111100110000111101101011110111000100110010010000100111111110001100011111111001000011001000101001

**H2(1 flip)-SHA512:** edf3eea01a79eb68531d6a64cc4bf988304c33a340d620f885d1812e3814ca79b64b0499949cd963384b3c8a2d7c6c20d490da40b2adbd4600b9199e0303ee1f

**Binary:** 11101101111100111110111010100000000110100111100111101011011010000101001100011101011010100110010011001100010010111111100110001000001100000100110000110011101000110100000011010110001000001111100010000101110100011000000100101110001110000001010011001010011110011011011001001011000001001001100110010100100111001101100101100011001110000100101100111100100010100010110101111100011011000010000011010100100100001101101001000000101100101010110110111101010001100000000010111001000110011001111000000011000000111110111000011111

**H2(1, 49, 73, and 113 flip)-SHA512:** c6948ea468c0b848cccd3cdd48a4acad0ff4dfca8de73bc8674b8d30589b2b3491cc448353a5e41b81b0cfdf2cccf59f5b8ec4e69bfcb57c54a216533868efcf

**Binary:** 11000110100101001000111010100100011010001100000010111000010010001100110011001101001111001101110101001000101001001010110010101101000011111111010011011111110010101000110111100111001110111100100001100111010010111000110100110000010110001001101100101011001101001001000111001100010001001000001101010011101001011110010000011011100000011011000011001111110111110010110011001100111101011001111101011011100011101100010011100110100110111111110010110101011111000101010010100010000101100101001100111000011010001110111111001111

For the SHA256 H1 and H2(1 flip), I got that they shared 131 bits out of 257. This is approximately 50%, which is what is to be expected from a good algorithm. When one bit is flipped, it should avalanche into about 50% of the total bits flipping in the result.

This trend continued regardless of which bit was flipped, or if multiple bits were flipped. For H2 (four bits flipped), the total shared also came out to about 50% also (123/257). This makes sense, too, as each bit change flips about 50% of the bits in the result. Do this with four bits, and with each 50% flip, some of the previously flipped bits (about 50% of those, or 25% overall) will flip back to being the same. Overall, this will average out to 50% of the bits being different with all the flipping.

The same trend was also seen with the SHA512 hashes, with each HS, when compared to H1, had about 50% different bits.