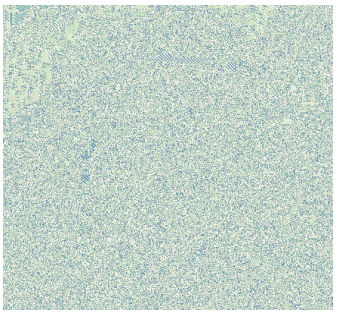
CS 370 Programming Project 1
Riley Ovenshire

Task 3.1: Encryption Variability

- **Plaintext**: "A screaming comes across the sky."
- **Ciphertext**:
    - *AES-256-ECB Output*: "eb14c0d34d490159993ba2d4dcfd0a98bf91383fd86d96066ae58213fb7e30e9f5af1ea5c734948022dd2dd4b99f26e0"
    - *Base64 Output:* "IkEgc2NyZWFtaW5nIGNvbWVzIGFjcm9zcyB0aGUgc2t5LiIgDQo="
    - *Blowfish-CBC Output:* "de95d628fe4c02c06030243b053e251f24d82363e86e165f1150a0ed441f6f484604b56d95486859"


Task 3.2: Image Encryption

- **Above***: Original Image -> ECB -> CBC*
- **Notes***:* I am unable to discern much from the CBC in its full resolution but here in the document I can at least see what I think is an outline of certain parts of Watto's snout. In the full resolution version of the ECB, I am actually able to see the outline of the hat.

Task 3.3: Role of Initialization Vectors

- **Do the contents of 'encrypt1.bin' match the contents of 'encrypt2.bin'? Explain why or why not.**
    - Yes, they match because they actually use the same plaintext, key and IV. When this is encrypted with the same scheme, the ciphers are the exact same.

- **Do the contents of 'encrypt1.bin' match the contents of 'encrypt3.bin'? Explain why or why not.**
    - No, changing the IV changes the result as well because the same plaintext is now encrypting to different ciphertexts under the same key.

Task 3.4: Encrypting with OpenSSL Crypto Library

- To compile: gcc -o main openssl-encryption.c -lcrypto
- Run ./main , everything will take care of itself in the program as long as the entire folder is present with words.txt, plaintext.txt and the C program.

Task 3.5: Message Digest Generation

- **Plaintext:** Used the same file as above - “This is a top secret.”
- **Algorithm:Digest -**
    - **SHA1**(plaintext.txt)= c8741dda267ab190ceb4c7c670e7c60fa846df6a
    - **SHA2-256**(plaintext.txt)= dfa187aaf85b7b15f502c6bb439641c93d4dbba9eb3fedb3f7e51e504323af9b
    - **SHA2-512**(plaintext.txt)= 599095d14e1ffb71e75b3da19343b5a12d8e92dc2163e9b9c7885e4b8739883236 adf3be31cab7e788fb40931148b0c45fbd4ecf350fda3bc97572e9b7fd036c

Task 3.6: Keyed Hash (HMAC)

- **Plaintext:** Used the same file as above - “This is a top secret.”
- **HMAC-SHA2-256**(plaintext.txt with key “*pynchon*”)= 0fb3d4190ea165aaa69994fab31a62f545329f83a97867a7df0998a1685e7a29
- **HMAC-SHA2-256**(plaintext.txt with key “*vollmann”*)= 01f55ef696e23aa5e432da2937a52d70f0ef16f5c4725c2ba2371db446aaad8e
- **HMAC-SHA2-256**(plaintext.txt with key *“delillo”*)= 28f2906af7349e427d8ad68cc52000942f5213f6d1cd377b4be37440403941a8
- **HMAC-SHA1**(plaintext.txt with key *“cambodia”*)= f1cafb4ca2435c10ef84ae912eb25afc524b7b43
- **HMAC-SHA1**(plaintext.txt with key “*greedo*”)= 2b040c730a08a9602c2e7ea1f3df6c29aac616c5
- **HMAC-SHA1**(plaintext.txt with key “*deliverance”*)= 0c79b7ce7d330fc4927632c097f0abca912b2fdc
- **Answer to question:** The key size in HMAC does not need to be fixed because the algorithm actually does it automatically by padding the key to fit the necessary length.

Task 3.7: Hash Function Randomness

- Roughly half of the bits of H1 and H2 are different from one another, it is surprising to see the amount of change that can be brought forth just by applying this change on a few bits.
- As more bits are flipped there is no way of predicting or telling which will be next. It seems like there is always at least half of the bits that are changed with each flip.
- When flipping 2 bits it seems like there is an exponential change in the hashing, and flipping 4 or more makes it even more intense with its differences.

Task 3.8: Collision Resistance Testing

Results:

riley@riley:~/Security/cs370/proj1/3.8 - collision$ ./main
Weak collision average trial count to find message: 7937634
Strong collision average trial count to find hash: 12116

riley@riley:~/Security/cs370/proj1/3.8 - collision$ ./main
Weak collision average trial count to find message: 9355621
Strong collision average trial count to find hash: 9085

riley@riley:~/Security/cs370/proj1/3.8 - collision$ ./main
Weak collision average trial count to find message: 19465778
Strong collision average trial count to find hash: 7487

riley@riley:~/Security/cs370/proj1/3.8 - collision$ ./main
Weak collision average trial count to find message: 8981286
Strong collision average trial count to find hash: 3160

riley@riley:~/Security/cs370/proj1/3.8 - collision$ ./main
Weak collision average trial count to find message: 7248225
Strong collision average trial count to find hash: 2866

ANSWERS:

- First off, this was super fun to implement - definitely one of the more fascinating projects to run throughout my time at OSU!
- Weak average: 10,597,709
- Strong average: 6,943
- It was significantly easier to break the strong collision, so much so that it even seemed that the program was getting better and better at finding collisions. I believe this is do to the fact that the discovered hashes are marked as “1” after being found (and this array never clears), so that significantly reduces the amount of iterations that the program has to do. It was a little surprising to see this reduce so quickly but it actually all made sense because of the Birthday Paradox. In just 5 attempts, reducing the needed amount of trials to 2,866 follows the principles presented in the paradox.
- The major differences that I encountered just pertain to the patterns between trial counts. It is pretty easy to predict that the strong collision is going to get better and better with each run, whereas the weak is relatively unpredictable. This definitely makes sense though because of the variability of characters that could go into a message. Super fascinating altogether and it really made stuff click for me. Loved this project!!!