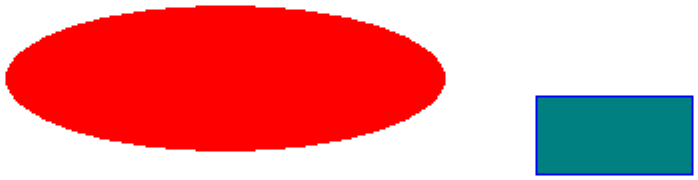
1. With CFB, the plaintext would not be recoverable given that during decryption, the cyphertext is XOR'd with the decrypted initialization vector. So if the 2nd bit of the 3rd block were corrupt, then any block after block 2, would be unrecoverable. Now with CTR mode, only the 3rd block would be unrecoverable, given that there is no chaining involved during its decryption process.
2. f(x) = x mod 10000 is not a good one-way hash function because it is easily predictable, in turn, making it easily reversable. This fails both properties required from a one-way has function. The x variable is easy to guess, and its easy to recreate. Hash functions should also avoid collisions, and using the value 10,000, so this function generates collisions for all values of x that differ by 10,000. So f(12345) and f(22345) both result in the value of 2345, resulting in collision. This function also forces uniformity only on the last few digits, making the distribution a bit skewed if the data is non-uniform.
3. The salt does not prevent attacks, it simply makes the brute force attacks harder to perform. If the salt were hidden or encrypted, you would need an extra mechanism to retrieve and decrypt the salt before password verification. If an attacker, compromises the shadow file, they are likely to compromise the salt mechanism as well. The salt is not meant to be a secret, it just makes the pre-computed attacks harder.
4. Sending the hashed password would not directly improve security if the hash itself is used as the password on the sever side. If the password is only hashed on the client side, and the hashed password is the password being matched on the server side, there would not necessarily be any difference between sending the hash versus plaintext given that the hash would effectively just become the password. An attacker who intercepts the hash, now has the direct password, hashed or not. This could be improved by including the hashing at the server side with salting.
5. By doing this, it makes brute force attacks expensive. By hashing a password many times, like 5000 rounds) each guess takes significantly longer to compute. This also increases the cost of attacking exponentially. For example, if a single hash takes 1 ms to compute, 5000 rounds would make this 5 seconds per guess.
6. SHA-256 is vulnerable to length-extension attacks. If an attacker knows h=SHA256(K || M), they can use it as the internal state to compute SHA256(K || X) for a new message X without knowing K. If the attacker makes X = M || N, then the attacker can calculate SHA256(K || M || N) which in turn equals SHA256(K || X).
7. Yes, the length of the key is important in a length-extension attack, but the exact length is not required, just an estimate. This is because the padding added to the input during the hash computation depends on the total length of the input, including the key.
8. a. Since the total length is 408 bits=(10*8)+(40x8)+(1*8) and the padding in SHA-256 is a multiple of 512. Since the last 64 bits are the length field, we need to get to 448 for the message field. 448-408=40, so 1 bit of 1 and 39 bits of value 0 will be added as padding.
   b. We need to first reconstruct the padding for K:M. From part a we found the padding. We append the padding to K:M as the hash state hash(K:M). We append the extra content N to K:M after the padding. N is going to be the padding and "extra content" added. Then we input N into the has function to compute hash(K :M :N).
9. Bob can use HMAC to securely verify Alice's identity without requiring her to reveal the secret number K. Bon sends Alice a randomly generated challenge message M. Alice

computes HMAC(K,M) using the shared secret K and sends the result back to Bob. Since only Alice and Bob know K, Bob can then independently compute HMAC(K, M) and compare it with Alice's response. If they match, Bon can confirm that he is communicating with Alice, since only someone who knows K could produce the correct HMAC value.

10. The Merkle-Damgard construction is used to design cryptographic hash functions that process variable-length inputs and produce fixed-length outputs. It works by dividing the input message into fixed-size blocks and processing each block sequentially using a compression function. The construction starts with an initialization vector, a fixed initial state. Each block is combined with the current state using the compression function, which outputs a new state. This process continues for all blocks, with the final state being the hash output. It also uses padding to make the input length a multiple of the block size and appends the message's total length to prevent extension attacks. This structure ensures that the hash is resistant to collisions and can handle variable-length inputs securely.
11. If bob had 2 messages: M1="HelloWorld123" and M2="HelloPlanet456", to create a colliding pair, Bob can append X = "ExtraContent" to both messaged. He constructs M3 = M1 || X = "HelloWorld123ExtraContent" and M4 = M2|| X = "HelloPlanet456ExtraContent". The hashes of M3 and M4 will also collide.

SECTION 2

1.



a.

Figure 1: Shapes Picture Original

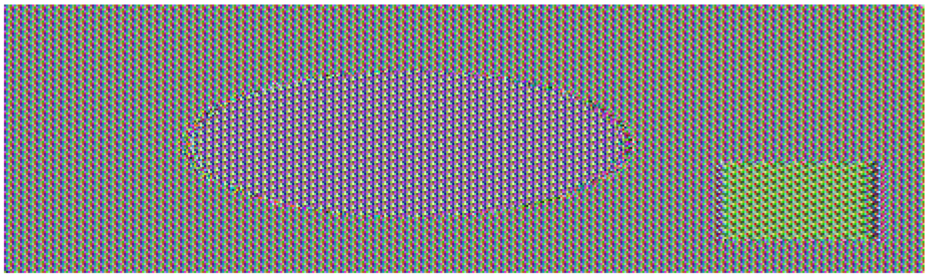


Figure 1: Shapes with ECB Encryption

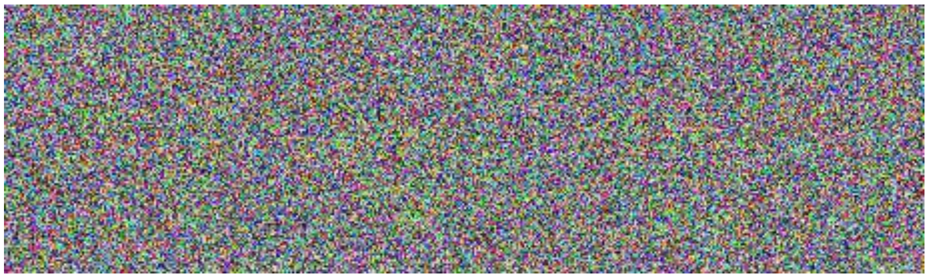


Figure 3: Shapes with CBC Encryption

b. In the image with ECB mode, the shapes are still somewhat visible. The shapes are distorted in color but the location of the color differences are in the same spots as the original image. This is because the ECB mode takes each block and encrypts them with the same key separately, so every block is uniformly encrypted, making it somewhat visible. As far as the image with CBC mode, the image is completely unreadable. Since each block is chained through the encryption process, this deepens the strength of the encryption. Rather than encrypting each block with the same key, in CBC each block is encrypted with the ciphertext of the previous block.

2.

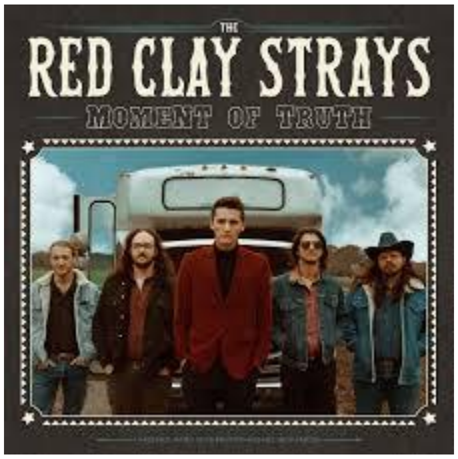


Figure 4: Red Clay Strays Original

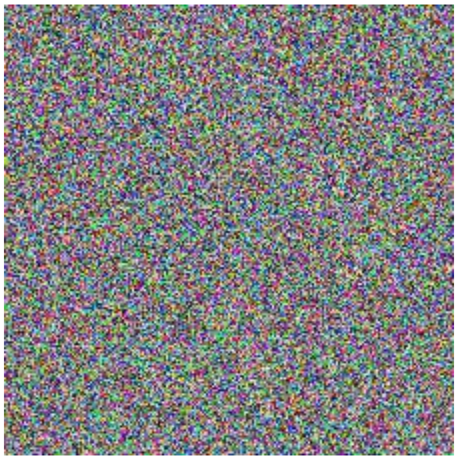


Figure 5: Red Clay Strays with ECB Encryption

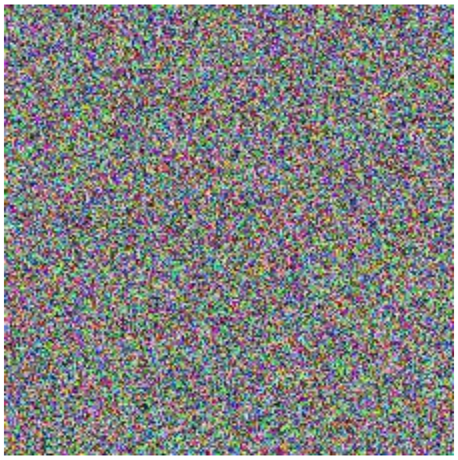


Figure 6: Red Clay Stays with CBC Encryption

I chose a picture from the Red Clay Strays album called Wondering Why. This image has script at the top and figures of people within it with a variety of colors, so I figured this would be a good image to test. As I can tell, using ECB versus CBC did not change much visually, as both are illegible. I think that this is likely because the lines and edges are softer than the image with shapes. In recognition algorithms like face, retina, and fingerprints, the edges represent the pixel color differences between one object and another. An picture like the shapes, shows strong edges, making it a bit resilient to EBC encryption. But an image like an album cover with many objects, visual designs, and color blending, would include edges that are simply just not as prominent. With that being said, both modes succeeding in encrypting my image to make it illegible from a human eye.