Title: Attacking Classic Crypto Systems

Omar faruk
Reg No: 2019831055

May 6, 2024

Task – 1: AES encryption using different modes

1 INTRODUCTION

In this project, we explore various encryption algorithms and modes, focusing on AES (Advanced Encryption Standard), a widely used symmetric encryption algorithm. Specifically, we investigate three different modes of AES encryption: CBC (Cipher Block Chaining), CFB (Cipher Feedback), and OFB (Output Feedback).

2 OBJECTIVE

1. To understand the concept of AES encryption.
2. To explore different modes of AES encryption.
3. To implement encryption and decryption using OpenSSL.

3 Implementation

- AES-128-CBC Mode:
  `openssl enc -aes-128-cbc -e -in input.txt -out output_cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

- AES-128-CFB Mode:
  `openssl enc -aes-128-cfb -e -in input.txt -out output_cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

- AES-128-OFB Mode:
  `openssl enc -aes-128-ofb -e -in input.txt -out output_ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

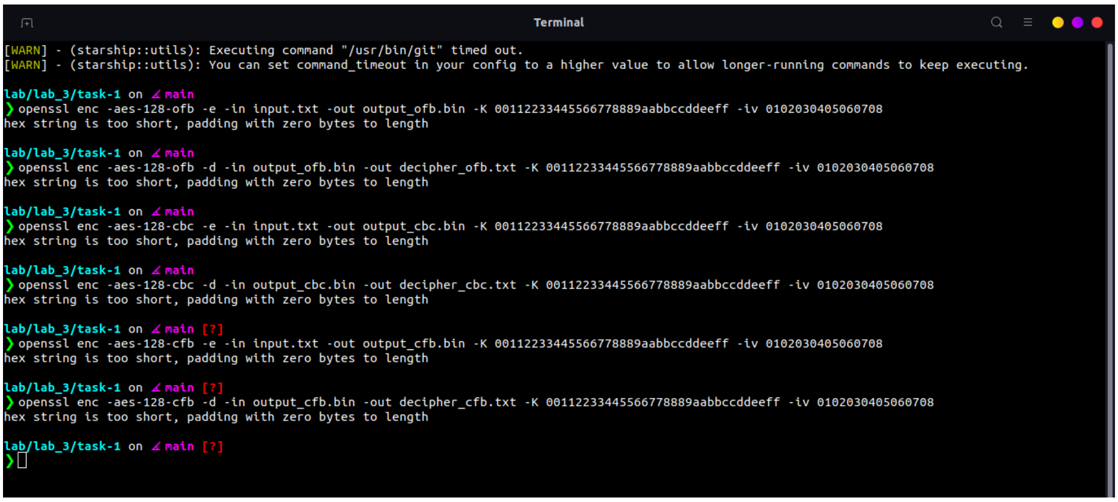


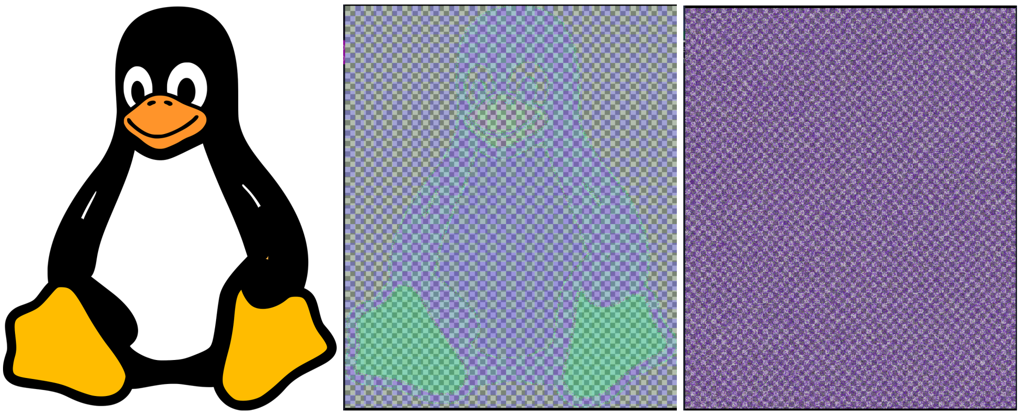
Figure 3.1: command for encryption and decryption

Task – 2: Encryption mode - ECB vs CBC

Encrypting using the ECB and CBC-

- Using OpenSSL with ECB mode:
  `openssl enc -aes-128-ecb -e -in original.bmp -out encrypted_ecb.bmp -K 00112233445566778889aabbccddeeff`

- Using OpenSSL with CBC mode:
  `openssl enc -aes-128-cbc -e -in original.bmp -out encrypted_cbc.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708`



(a) Original image (b) ECB encrypted image (c) CBC ebcrypted image

Figure 3.2: Original and after encryption image

OBSERVATION

The ECB encrypted image might exibit patterns or repetation, especially if the image contains repetitive patterns.This is because ECB mode encrypt identical plaintex blocks into identical ciphertext blocks ,which can expose patterns in the image.

On the other hand ,the CBC encrypted image is less likely to exhibit such patterns because each block of plaintex is XORed with previous ciphertex block before encryption ,adding randomness to the encryption process.

Task – 4: Padding

Experimental Procedure

The plaintext file plaintext.txt was encrypted using OpenSSL with ECB, CBC, CFB, and OFB modes.

- ECB Mode::
  `openssl enc -aes-128-ecb -e -in plaintext.txt -out encrypted_ecb.txt -K 00112233445566778889aabbccddeeff`
- CBC Mode:
  `openssl enc -aes-128-cbc -e -in plaintext.txt -out encrypted_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708`
- CFB Mode:
  `openssl enc -aes-128-cfb -e -in plaintext.txt -out encrypted_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708`
- OFB Mode:
  `openssl enc -aes-128-ofb -e -in plaintext.txt -out encrypted_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708`

Table 3.1: Padding Requirements for Different Encryption Modes

Encryption Mode	**Padding Required**	**Explanation**
ECB	No	ECB mode operates on fixed-size blocks and does not require padding. Each block is encrypted independently, making padding unnecessary.
CBC	Yes	CBC mode requires padding because it operates in chained blocks and the final block may not always be full. Padding ensures that the plaintext is a multiple of the block size.
CFB	No	CFB mode does not require padding because it operates on a block-by-block basis, where each block is used to encrypt the next block of plaintext. Padding is not needed as the plaintext is processed in units equal to the block size.
OFB	No	Similar to CFB mode, OFB mode does not require padding because it generates a keystream independent of the plaintext size. Each block of the plaintext is XORed with the corresponding block of the keystream.

Task – 5: Generating message digest

Table 3.2: Hash Values Generated Using Different Algorithms

Command	Hash Value
`openssl dgst -md5 message.txt > md5_hash.txt`	MD5(plaintext.txt)= c0cf81e6fc699c60b687b63b249ae3af
`openssl dgst -sha1 message.txt > sha1_hash.txt`	SHA1(plaintext.txt)= 32814bf55a3f3a2ee3492961df59dc0593bd9f3b
`openssl dgst -sha256 message.txt > sha256_hash.txt`	SHA256(plaintext.txt)= 7f1c437c78909095f6b7d36e7f98c86172aa9586e49a6a13f1a0f26f5f4824fd

Task – 6: Keyed hash and HMAC

Table 3.3: HMAC Values Generated Using Different Algorithms

Algorithm	Command	HMAC
HMAC-MD5	`openssl dgst -md5 -hmac "key" message.txt`	e48d2535b84d3c8b3b3c6d5fc6f367ee
HMAC-SHA256	`openssl dgst -sha256 -hmac "long_secret_key" message.txt`	db3c2d70c1c3688d5d19b7734d96c1b6b1e6b7228e0325e00030df628cb285f3
HMAC-SHA1	`openssl dgst -sha1 -hmac "12345678901234567890" message.txt`	5e30192fc7c49d57a9611a3a3d9cfb27be1f9243

Figure 3.3: Commands

Task – 7: Keyed hash and HMAC

Table 3.4: HMAC Values Generated Using Different Algorithms

Algorithm	Command	HMAC
MD5(H1)	`openssl dgst -md5 plaintext.txt`	c0cf81e6fc699c60b687b63b249ae3af
SHA256(H1)	`openssl dgst -sha256 plaintext.txt`	7f1c437c78909095f6b7d36e7f98c86172aa9586e49a6a13f1a0f26f5f4824fd
MD5(H2)	`openssl dgst -md5 plaintext.txt`	2ca248db83121b9dbee0e26819298736
SHA256(H2)	`openssl dgst -sha256 plaintext.txt`	770e5e38708c501aa919bf41f894c51823c032a370e7f33a654bab7d85299e3f

Observation

If the hash values H1 and H2 are not the same before and after the bit flipping, it indicates that even a small change in the input data can lead to a significant difference in the resulting hash values.

Listing 1: number of same bits counting

```
#include <iostream>
#include <string>
#include <bitset>
using namespace std;
int countSameBits(const string& hash1, const string& hash2) {
    int count = 0;
    for (size_t i = 0; i < hash1.length(); ++i) {
        bitset<8> bits1(hash1[i]);
        bitset<8> bits2(hash2[i]);
        count += (bits1 ^ bits2).count();
    }
    return count;
}

int main() {
    string H1 = "c0cf81e6fc699c60b687b63b249ae3af";
    string H2 = "2ca248db83121b9dbee0e26819298736";

    int sameBitsCount = countSameBits(H1, H2);

    cout << "Number of same bits between H1 and H2: " <<
       sameBitsCount << endl;

    return 0;
}
```

Output: Number of same bits between H1 and H2: 85